НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 519.254

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2019 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 124 Системний аналіз
на тему: «Система торгівлі цінними паперами методами навчання з підкріпленням»

Виконав:
студент ІІ курсу, групи КА-82 мп
Мороз Андрій Ярославович _____

Керівник: в.о.завідувача кафедри ММСА, к.т.н., доц.
Тимощук Оксана Леонідівна _____

Рецензент: доцент кафедри міжнародної економіки
КПІ ім. Ігоря Сікорського к.е.н., доц.
Савченко Сергій Миколайович _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 124 «Системний аналіз»

Затверджую
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Морозу Андрію Ярославовичу

1. Тема дисертації: «Система торгівлі цінними паперами методами навчання з підкріпленням», науковий керівник дисертації, к.т.н., Тимощук О.Л., затверджені наказом по університету від
«___» _____ № _____
2. Термін подання студентом дисертації: 13 грудня 2019 р.
3. Об’єкт дослідження: набір даних щодо операцій купівлі та продажу цінних паперів на біржі
4. Предмет дослідження: навчання з підкріпленням в фінансових ринках
5. Перелік завдань, які потрібно розробити:
   а) дослідити сучасний фінансових ринків;
   б) ознайомитись з сучасними методами алгоритмічної торгівлі;
   в) порівняти класичні методи алгоритмічної торгівлі з навчанням з підкріпленням;
   г) дослідити використання навчання з підкріпленням до фінансових ринків;
   д) застосувати навчання з підкріпленням до алгоритмічної торгівлі;

е) розробити стартап-проект;

ж) розробити концептуальні висновки.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

а) Схема форматів методу (рис.);

б) Приклади функціонування створеного програмного продукту (рис.);

в) Таблиці у розділі стартап-проекту

7. Дата видачі завдання: 05 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1.	Концептуальний вступ дисертації. Формулювання об'єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	18.09.2019—20.09.2019
2.	Перший розділ. Аналіз методів навчання з підкріпленням	21.09.2019—30.09.2019
3.	Другий розділ. Аналіз процесів алгоритмічної торгівлі методами навчання з підкріпленням	31.09.2019—16.10.2019
4.	Третій розділ. Проведення експерименту	17.10.2019—25.10.2019
5.	Четвертий розділ. Стартап-проект	03.11.2019—06.11.2019
6.	Концептуальні висновки. Перспективи розвитку отриманих рішень	07.11.2019—10.11.2019

Студент А. Я. Мороз

Науковий керівник дисертації О. Л. Тимощук

РЕФЕРАТ

Магістерська дисертація: 87 с., 22 табл., 19 рис., 1 додаток, 18 джерел.

Актуальність теми: в світі бурхливо зростає популярність нових підходів до алгоритмічної торгівлі. Проте, разом з цим, зростає і кількість трейдерів, основною задачею яких є одержання прибутку. Таким чином, розробка та застосування систем торгівлі у процесі прийняття рішення щодо здійснення операцій купівлі продажу цінних паперів є актуальною на сьогоднішній день.

Мета даної роботи полягає у дослідженні та вдосконаленні існуючих методик побудови моделей навчання з підкріпленням для торгівлі цінними паперами та розробці системи що здійснюватиме цю торгівлю.

Об’єктом дослідження є набір статистичних даних щодо операцій купівлі та продажу цінних паперів на біржі.

Методи дослідження: моделі навчання з підкріпленням, нейроннні мережі та операції над матрицями.

Програмний продукт реалізований за допомогою мови програмування Python 3.7 у середовищі розробки Jupyter Notebook.

Отримані результати: розроблено систему систему торгівлі цінними паперами за допомогою навчання з підкріпленням. Проведено апробацію програмного продукту на реальних даних.

ФІНАНСОВИЙ РИНОК, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ТРЕЙДИНГ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, АЛГОРИТМІЧНА ТОРГІВЛЯ.

ABSTRACT

Master’s thesis explanatory note: 87 p., 22 tabl., 19 fig.,1 application, 18 references.

Relevance of the topic: new approaches to algorithmic trading are growing rapidly in the world. However, at the same time, there is an increasing number of traders whose main task is to make a profit. Thus, the development and application of trading systems in the decision-making process for the sale of securities is currently relevant.

The purpose of this work is to research and improve existing methods of building models of reinforcement training for trading securities and developing a system that will carry out this trade.

The object of the study is a set of statistics on the operations of buying and selling securities on a stock exchange.

Research Methods: Supportive learning models, neural networks, and matrix operations.

The software is implemented using Python 3.7 programming language in the Jupyter Notebook development environment.

Results obtained: Securities trading system was developed using reinforcement training. The software is tested on real data.

FINANCIAL MARKET, MACHINE LEARNING, NEURAL NETWORKS, TRADING, REINFORCEMENT LEARNING, ALGORITHMIC TRADING.

ЗМІСТ

ВСТУП

Алгоритмічна торгівля існує десятиліттями і здебільшого користується неабияким успіхом у різноманітних формах. Традиційно алгоритмічна торгівля включає набір правил торгівлі, які ретельно розроблені, оптимізовані та перевірені людьми. Незважаючи на те, що ці стратегії мають перевагу в тому, що вони є систематичними та здатними працювати зі швидкістю та частотою, що перевищує торговців-людей, вони чутливі до будь-яких упереджень вибору та не в змозі адаптуватися до мінливих ринкових умов.

З іншого боку, навчання з підкріпленням набагато більш вільне. У навчанні з підкріпленням агент просто прагне досягти максимуму своєї винагороди в будь-якому даному середовищі і намагається вдосконалити прийняття рішень шляхом спроб та помилок, оскільки вивчає багато прикладів. Він також може навчитися приймати рішення, базуючись не лише на своїх переконаннях про навколишнє середовище на крок вперед, але і на тому, як ринок розвивається далі. У більшості традиційних алгоритмів торгівлі є окремі процеси прогнозування, перетворюючи це прогнозування на дію та визначаючи частоту дії на основі транзакційних витрат. Навчання з підкріпленням підтримує підхід, який інтегрує ці процеси. З усіх цих причин навчання з підкріпленням може виявити дії, яких люди зазвичай не знаходять.

Ціллю даної роботи є дослідження моделей навчання з підкріпленням в фінансовій сфері та їх використання для торгівлі цінними паперами в фінансових установах. Для досягнення даної мети були вирішені наступні задачі:

а) проаналізовано існуючі рішення для алгоритмічної торгівлі;
б) проведено огляд та аналіз існуючих математичних методів моделювання і алгоритмічної торгівлі;
в) розроблено архітектуру системи торгівлі цінними паперами..

Практичним результатом роботи є модель, яка дозволяє здійснювати автоматичну торгівлю цінними паперами.

Робота складається з 4 розділів. В першому розділі розглядаються поняття та сутність навчання з підкріпленням. У другому розділі вивчаються підходи та методи використання навчання з підкріпленням в алгоритмічній

торгівлі. У третьому розділі дисертації описується розроблена модель, а також проводиться порівняння результатів роботи системи з іншими методами. Четвертий розділ присвячено розробленню стартап-проекту на основі створеного програмного продукту.

РОЗДІЛ 1
НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Найбільш популярними підходами до машинного навчання є навчання з учителем і навчання без учителя. Але чи так люди навчаються? Проблема у тому, що ми далеко не завжди знаємо правильні відповіді. Часто ми просто робимо ту чи іншу дію і отримуємо результат. Коли дитина вчиться ходити, вона спочатку робить щось віддалено схоже на правду, що підказує йому інстинкт, і отримує результат, спочатку швидше за все негативний. Тоді вона трохи змінює свою поведінку, можливо, досить випадковим чином, і дивиться, чи не збільшилася чи, так би мовити, цільова функція; а коли щось починає виходити, дитина запам'ятовує, як це сталося, і потім повторює те саме, намагаючись розвинути успіх далі.

Звідси і основна ідея навчання з підкріпленням (reinforcement learning). У цій постановці завдання агент взаємодіє з навколишнім середовищем, виконуючи дії; довкілля його заохочує за ці дії, а агент продовжує їх робити, намагаючись максимізувати свою «нагороду» за це; його нагорода теж приходить з навколишнього середовища.

Історію навчання з підкріпленням можна починати від робіт Павлова про умовні і безумовні рефлекси, хоча психологічні підходи були відомі і раніше, наприклад в роботах Олександра Бена середини XIX століття [1]. Його основна ідея полягала в тому, що ми навчаємося методом проб і помилок: коли якийсь спонтанний (тобто по суті випадковий) рух збігається з почуттям задоволення, «утримує сила духу» встановлює між ними асоціацію. Ту ж лінію продовжили і теорії Ллойда Моргана в психології і Едварда Лі Торндайка в його працях про інтелект тварин [2]: корисна дія, що викликає задоволення, закріплюється і підсилює зв'язок між ситуацією і реакцією, а шкідливий, що викликає незадоволення, послаблює зв'язок і зникає. А після того як в психологію прийшли біхевіористи, і особливо Б. Ф. Скіннер, ця ідея стала домінуючою.

В цілому це і є ідея навчання з підкріпленням, і в машинному навчанні

вона теж з'явилася дуже давно. Ще Тьюринг в своїх есе про штучнийе інтелект описував архітектуру системи «болю і насолоди» яка повинна керувати тим, що саме зберігається в пам'яті штучного інтелекту [3]. Ранні роботи інформатиків на цю тему схожі на роботи Павлова і Скіннера: наприклад, в 1952 році Клод Шеннон продемонстрував лабіринт, по якому бігало мишеня на ім'я Тесей, досліджуючи його тим самим методом проб і помилок і запам'ятовуючи свій шлях [4].

У своїй дисертації Марвін Мінський [5] представив обчислювальні моделі навчання з підкріпленням, а також описав аналогову обчислювальну машину, побудовану на елементах, які він назвав SNARC - стохастичний обчислювач, якого навчають через підкріплення і по суті аналогічний нейрону. Ці елементи повинні були відповідати змінним семантичним зв'язкам в людському мозку.

Висловлюючись більш формально, на кожному кроці агент може знаходитись в деякому стані $s \in S$, де S — множина всіх станів, і обирає деяку дію $a \in A$ з множини можливих дій A.

Після цього довкілля повідомляє агенту, яку нагороду r (від слова reward) він за це отримав і в якому стані s' опинився в результаті своїх дій. Завдання агента — заробити якомога більшу нагороду:

- або за обмежений час h; така постановка називається моделлю з скінченим горизонтом і цільову функцію можна записати так:

$$R = \mathbb{E}\left[r_0 + r_1 + r_2 + ... + r_h\right] = \mathbb{E}\sum_{t=0}^{h} r_t,$$

  де r_t — винагорода на кроці t;
- або за весь нескінченний майбутній час; в такому випадку, звичайно, просто підсумовувати не вийде, але легко зрозуміти, що в таких завданнях майже завжди отримати нагороду раніше вигідніше, ніж пізніше, тому зазвичай в цільову функцію моделі з нескінченним горизонтом вводять деяку константу, на яку винагорода зменшується з кожним наступним кроком:

$$R = \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^k r_k + ...\right] = \mathbb{E}\sum_{t=0}^{\infty} \gamma^t r_t.$$

1.1 Multi-armed bandits

Найпростіша постановка задачі навчання з підкріпленням — це так зване завдання про багаторуких бандитів (multiarmed bandits). Формально тут все точно так само, але $|S| = 1$, тобто стан агента не змінюється. У нього просто є якийсь фіксований набір дій A і можливість вибирати з цього набору дій. Така модель називається завданням про багаторуких бандитів, тому що її легко уявити собі так: агент знаходиться в кімнаті з декількома ігровими автоматами, у кожного автомата своє очікування виграшу, а агенту потрібно виграти якомога більше грошей, кидаючи монетки то в один автомат, то в інший.

Виходить, що агент сам платить за своє навчання, і йому потрібно зуміти вчасно зрозуміти, що навчання (дослідження, exploration) можна закінчувати і переходити до використання отриманих знань (exploitation). Дослідження проти експлуатації — це головна проблема, основна дилема завдання про багаторуких бандитів.

У цій роботі не буде детально розглянуто багаторуких бандитів, тому що глибоке навчання з підкріпленням зазвичай починається там, де станів кілька, але пару важливих зауважень зробити потрібно. Почнемо з найпростішого жадібного алгоритму, який завжди вибирає стратегію, що максимізує прибуток. Прибуток можна оцінити як середня винагорода, отримана від дії:

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_{k_a}}{k_a}.$$

Жадібний алгоритм здається досить простим і логічним. Що ж з ним не так? Справа в тому, що оптимум легко прогледіти, якщо на початковій вибірці нам трошки не пощастить, що більш ніж можливо. Уявіть собі, наприклад,

бінарних бандитів, у яких виплати завжди дорівнюють нулю або одиниці. Тоді жадібний алгоритм буквально ніколи не повернеться до тих ручок, які на першому обході видали нуль, адже їх середнє буде нульовим, а у тих ручок, які хоч раз видали одиницю, нуля вже ніколи не вийде.

Тому корисна евристика в завданні про багаторуких бандитів, та й в усьому навчанні з підкріпленням — оптимізм при невизначеності. Це означає, що обирати потрібно жадібно, але при цьому прибуток оцінювати оптимістично, і завжди вимагати серйозні свідчення, щоб перестати перевіряти ту чи іншу стратегію.

Одним з ключових теоретичних інструментів в навчанні з підкріпленням є ε-жадібна стратегія: вибрати дію з найкращим очікуваним прибутком з ймовірністю $1 - \varepsilon$, а з ймовірністю ε вибрати випадкову дію. Така стратегія призводить до того, що алгоритм не відрізняє хорошу альтернативу від непотрібної, виділяючи тільки кращу, але все одно процес дослідження виходить розумним, і про нього зазвичай можна довести різні корисні теоретичні властивості, адже ε-жадібна стратегія означає, що ми завжди можемо смикнути за кожну ручку з позитивною константної ймовірністю. На практиці зазвичай починають з великих ε, а потім зменшують; вибір стратегії цього зменшення — важливий параметр алгоритму.

Інший природний спосіб застосувати оптимістично-жадібний метод — це відомі з статистики довірчі інтервали. Давайте для кожної дії зберігати статистику числа таких дій n і числа успішних дій w, а потім для вибору ручки, за яку хочемо смикнути, використовувати верхню межу довірчого інтервалу ймовірності успіху (або очікування винагороди). Тим самим ми досягнемо якраз потрібного ефекту: спочатку всі довірчі інтервали будуть дуже широкими, і їх верхні межі будуть дуже високо, а потім, у міру накопичення досвіду, інтервали почнуть звужуватися, причому менш досліджені альтернативи будуть отримувати перевагу в виборі. Така стратегія зійдеться до оптимальної, коли довірчі інтервали всіх інших ручок будуть повністю лежати нижче її середнього.

Інший, більш простий варіант оптимізму при невизначеності — почати з оптимістичних значень середніх: давайте виставимо $Q_0(a)$ таким великим,

що будь-яка винагорода буде «розчаровувати» нас, але не занадто великими — нам потрібно, щоб досить швидко Q_0 усереднилась з реальними оцінками середніх. Тоді можна використовувати тривіальну жадібну стратегію, і вона дасть той же ефект: спочатку середні у всіх ручок будуть свідомо занадто високими, а потім у міру накопичення досвіду зменшаться і будуть поступово сходитись до істинних середніх, причому чим менше було експериментів, тим більше впливає Q_0 на середнє. Самі значення Q_0 можна міняти і тим самим управляти балансом між дослідженням і експлуатацією.

Сучасні алгоритми діють приблизно за тією ж загальною схемою: вони надають пріоритет кожної ручці і доводять оцінки безпосередньо на ціну навчання. Так, стратегія UCB1 [6] враховує невизначеність, що лишилась в тій чи іншій ручці, і намагається обмежити ціну навчання так: якщо з n експериментів n_i раз смикнули за i ручку і отримали середню нагороду $\hat{\mu}_i$, алгоритм UCB1 привласнює їй пріоритет

$$Priority_i = \hat{\mu}_i + \sqrt{\frac{2\log n}{n_i}}.$$

Також зауважимо як перераховувати оцінки середнього $Q_t(a)$:

$$Q_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} r_i = \frac{1}{k+1}\left[r_{k+1} + \sum_{i=1}^{k} r_i\right] = \\ = \frac{1}{k+1}(r_{k+1} + kQ_k) = Q_k + \frac{1}{k+1}(r_{k+1} - Q_k).$$

Вийшла дуже важлива формула, частковий випадок загального правила — зміщуємо оцінку так, щоб зменшувалась помилка:

$$NewEstimate = OldEstimate + Step[Target - OldEstimate].$$

Саме так виглядає, наприклад, загальне правило градієнтного спуску: похідна вказує напрямок на ціль в поточній точці, а крок — це швидкість навчання.

Помітно, що у середнього крок не постійний, а зменшується з часом: з формули Q_{k+1} виходить, що крок руки a в момент k дорівнює $(k_a+1)^{-1}$. Змінюючи послідовність кроків, можна домогтися інших ефектів. Наприклад, часто буває, що нагороди від різних ручок насправді нестаціонарні, тобто змінюються із часом. У такій ситуації має сенс давати великі ваги свіжій інформації, а далекому минулого — маленькі. Як це зробити? Можна просто замість коефіцієнтів, що затухають поставити постійні: у правила оновлення $Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$ з постійним $\alpha_k(a) = \alpha$ коефіцієнти будуть затухати експоненційно:

$$\begin{aligned} Q_k &= Q_{k-1} + \alpha[r_k - Q_{k-1}] = \alpha r_k + (1-\alpha)Q_{k-1} = \\ &= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} = \\ &= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i. \end{aligned}$$

Таке правило оновлення не сходиться (різниця між сусідніми Q_k і Q_{k-1} не обов'язково прагнуть до нуля зі зростанням k), але це і добре — ми хочемо слідувати за метою. Є й загальний результат: правило оновлення сходиться, якщо послідовність коефіцієнтів не сходиться сама, але сходиться сума квадратів коефіцієнтів.

1.2 Марковські процеси прийняття рішень

Тепер, коли ми зрозуміли основну суть навчання з підкріпленням і розібралися з найпростішою ситуацією, пора узагальнювати далі. В житті, звичайно, часто буває, що агент не повертається в точно такий ж стан для нового «ходу»: наприклад, граючи в го або шахи програма повинна, мабуть, враховувати, що позиція на дошці після її ходу і відповіді противника дещо

зміниться. Тому тепер нам потрібно визначити якийсь процес, в якому агент послідовно переходить із стану в стан в залежності від своїх дій, іноді, в деяких станах, отримуючи нагороди; всі залежності тут, звичайно, будуть не прямими, а стохастичними.

Тут виникає друга центральна дилема навчання з підкріпленням, задача розподілу винагороди (credit assignment): нехай навіть ми знаємо, виграли ми партію або програли, але який саме хід привів до перемоги або до поразки? Ця проблема теж була очевидна з перших кроків теорії навчання з підкріпленням, її розглядав ще Марвін Мінський на початку 1960-х років [7], але успішно розв'язати цю проблему буває складно досі.

Введемо основні поняття. Марковський процес прийняття рішень [8, 9] складається з:

- множини станів S;
- множини дій A;
- функції нагороди $R : S \times A \to \mathbb{R}$; це означає, що очікувана винагорода при переході із стану s в стан s' дорівнює $R^a_{ss'}$;
- фунцкії переходу між станами $p^a_{ss'} : S \times A \to \Pi(S)$, де $\Pi(S)$ — множина розподілів ймовірностей над S; це означає, що ймовірність попасти з стану s в стан s' після дії a дорівнює $p^a_{ss'}$;.

На рис. 1.1 проілюстровано марковський процес прийняття рішень: зліва, на рис. 1.1, а, ви бачите саму загальну, класичну схему навчання з підкріпленням. А на рис. 1.1, б показана більш детальна схема марковського процесу прийняття рішень, де видно, які його частини від яких залежать.

Процес називається марковським, тому що ймовірності переходів між станами не залежать від історії попередніх переходів; взагалі, слово «марковський» в математиці та інформатиці завжди позначає саме це: відсутність пам'яті, незалежність від того, що було в минулому. Це здається дуже сильним припущенням, але насправді воно досить часто виконується. Наприклад, в го або шахах неважливо, якими ходами ми прийшли до поточної позиції, важлива лише позиція сама по собі.

А якщо марковську властивість істотно порушено, то часто можна просто записати те, що потрібно «пам'ятати» з минулого, як частини визначення

стану, і тоді марковську властивість буде відновлено. Наприклад, стан при грі в покер має включати в себе не тільки поточний розмір ставок, але і історію ставок в поточній роздачі, а можливо, і більш довгу історію взаємодії між гравцями.

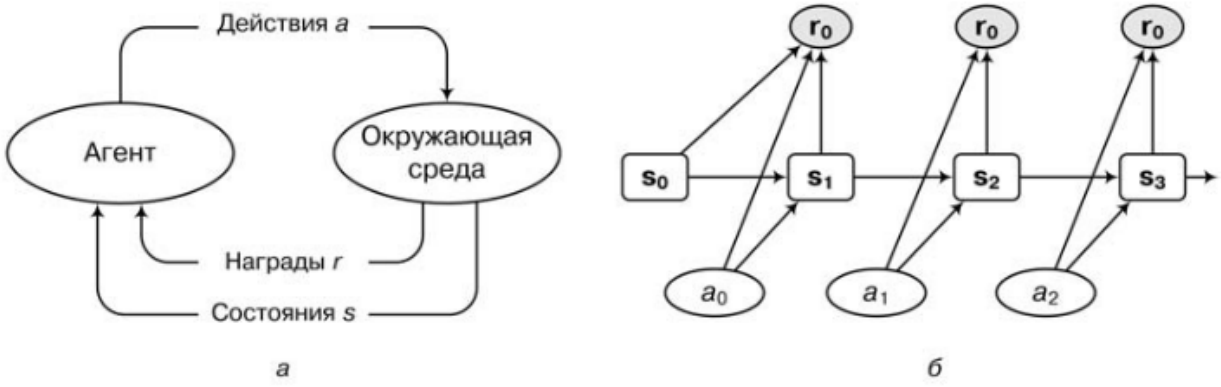

Рисунок 1.1 — Марківський процес прийняття рішень: а — загальна схема навчання з підкріпленням, б —детальна схема марковського процесу прийняття рішень

Тут варто зробити невеличкий відступ. Загальна постановка задачі навчання з підкріпленням схожа на постановку задачі теорії оптимального керування, в якому відома деяка модель об’єкта управління, на об’єкт є деякі важелі впливу, і завдання полягає в тому, щоб знайти оптимальні дії, які максимізують потрібну цільову функцію. В теорії оптимального керування з’явилися і рівняння Беллмана, про які йтиметься нижче, і принцип максимуму Понтрягіна. І об’єкти в ній розглядаються куди складніші, ніж зазвичай в навчанні з підкріпленням. Дійсно, багато чого з того, про що ми будемо говорити в цьому розділі, можна розглядати як частину теорії управління. Однак акценти розставлені дещо інакше: в навчанні з підкріпленням основна складність не в тому, щоб знайти оптимальне управління, а в тому, щоб вивчити навколишнє середовище — якщо ми середу і об’єкт управління вже знаємо (в теорії оптимального управління це називається ідентифікацією системи), зазвичай знайти оптимальне дію не так вже й складно.

Докладний приклад марковського процесу прийняття рішень, до якого

ми будемо постійно повертатися, наведено на рис. 1.2. На малюнку прямокутники відповідають станам, білі овали — можливим діям, а овали з штрихуванням — винагород, одержуваних на цьому переході між станами. Сам процес відповідає простій грі «парне-непарне», що проходить в два кола. Правила гри такі:

- в кожному колі і ми, і противник вибираємо число; якщо обидва числа парні або обидва непарні, ми виграємо; якщо парність різна (одне число парне, інше непарне), ми програємо;
- в першому колі виграш і програш дорівнює +1 і -1 відповідно, а у другому колі ставки підвищуються, і виграш і програш починають коштувати +5 і -5 відповідно;
- після другого кола гра закінчується;
- оскільки гра кінцева, процес виходить епізодичний, і $\gamma = 1$s.

Але це ще не все; щоб повністю поставити марковський процес прийняття рішень, потрібно ще визначити ймовірності переходів між станами; вони показані на стрілках, що відповідають переходам. Тут буде невелике ускладнення, без якого гра була б зовсім нудною. Противник буде не просто підкидати монетку, а діяти таким чином:

- першому колі противнику більше подобаються непарні числа: ймовірність парного числа від нього дорівнює 1/3, а непарного — 2/3;
- а в другому колі він дивиться на те, до чого призвела його гра в перший раз: якщо він в першому колі виграв, то він вважає, що зіграв правильно, і повторює свій вибір, а якщо програв, повертається до стратегії за замовчуванням і в другому колі просто підкидає монетку.;

Тепер, коли у нас є стани і переходи між ними, доведеться навчитися розрізняти функцію винагороди (reward function, безпосереднє підкріплення, те, що ми позначили за R) і те, що ми назвемо функцією значення стану (value function, $V(s)$); це буде загальне очікуване підкріплення, яке можна отримати, якщо почати з цього стану. Суть багатьох методів навчання з підкріпленням — в тому, щоб оцінювати і оптимізувати функцію значень; по суті завдання наше зводиться до того, щоб вибирати ходи, які призводять до стану з максимальним значенням $V(s)$. Для марковських процесів можна

формально визначити:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | s_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right],$$

де π — стратегія, якій слідує агент.

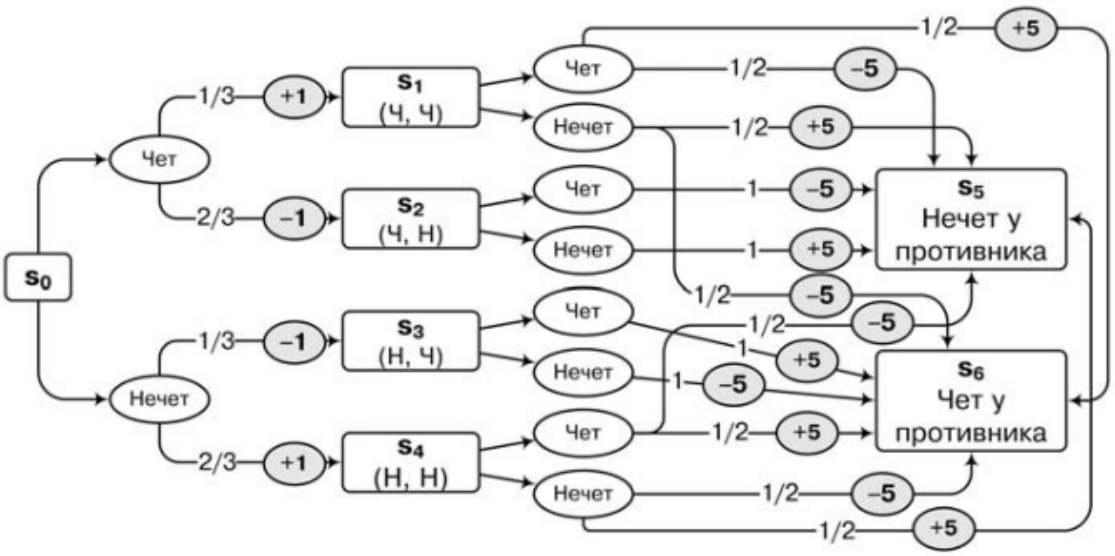

Рисунок 1.2 — Приклад марковського процесу прийняття рішень: гра «парне-непарне»

Зверніть увагу, що слово стратегія (policy) тут розуміється в строгому сенсі. Оскільки, взагалі кажучи, агент може підкидати монетки, щоб вибирати чергові дії, стратегія π — це функція, яка для даного стану s видає розподіл ймовірностей на множині дій A. Ми також будемо позначати через $\pi(a, s)$ ймовірність вибрати дію $a \in A$ в стані s, а якщо захочемо підкреслити, що стратегія задана параметрично з вектором параметрів θ, будемо писати $\pi(a, s, \theta)$. Якщо ж стратегія детермінована, це просто означає, що для кожного s всі ймовірності $\pi(a, s)$ дорівнюють нулю, крім однієї, яка дорівнює одиниці.

Втім, функція значень стану все ще віддалена від безпосередніх рішень агента, адже він не може просто взяти і вибрати наступний стан. Гравець в

го не може сам визначити позицію перед своїм наступним ходом, вона буде залежати і від ходу противника. Тому очікуване в майбутньому підкріплення часто розглядають більш детально: функція Q висловлює загальне підкріплення, очікуване, якщо агент почне в стані s і зробить там дію a:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t|s_t = s, a_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right].$$

Функції V і Q — це якраз те, що нам потрібно оцінити; якби ми їх знали, можна було б просто вибирати ту дію a, яке максимізує $Q(s,a)$.

Давайте спробуємо підрахувати функцію значень стану $V^{\pi}(s)$, послідовно розгортаючи її визначення. У ланцюжку рівностей нижче ми спочатку виписуємо визначення очікуваної сумарної винагороди з нескінченним горизонтом, потім відокремлюємо від нього перший крок і помічаємо, що після нього залишається точно такий ж вираз, просто помножений на γ:

$$\begin{aligned}
V^{\pi}(s) = \mathbb{E}_{\pi}[R_t|s_t = s] &= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right] = \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\right] = \\
&= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left(R^a_{ss'} + \gamma\mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_{t+1} = s'\right]\right) = \\
&= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V^{\pi}(s')).
\end{aligned}$$

У нас вийшло, що для відомої стратегії π значення V^{π} задовольняють так званим рівнянням Беллмана.

Рівняння Беллмана — це по суті математичний вираз принципу динамічного програмування. Такі рівняння і їх аналоги з'являються у великій кількості різних додатків. А в нашому випадку виходить, що, теоретично кажучи, для того щоб знайти значення $V^{\pi}(s)$, можна просто взяти і вирішити систему лінійних рівнянь, невідомими в яких є $V^{\pi}(s)$ для різних станів $s \in S$.

Правда, для цього потрібно знати всі параметри марковського процесу, тобто функції R і P, а з цим в реальних ситуаціях погано. Все, що у нас зазвичай є на вході, — це навколишнє середовище, що видає нагороди як чорний ящик.

Так що в реальних задачах функції R і P теж доводиться навчати по ходу справи. Насправді методи навчання з підкріпленням діляться на ті, які навчають функції R і P в явному вигляді, і ті, які обходяться без цього і відразу навчають V або Q.

Але для простих прикладів скористатися рівняннями Беллмана цілком можливо. Давайте спробуємо підрахувати функцію значень стану для якоїсь стратегії в прикладі на рис. 1.2. Наприклад, нехай стратегія π — це просто випадковий вибір на кожному кроці, підкидання чесної монетки. Тоді:

$$V^\pi(s_0) = \frac{1}{2}\left(\frac{1}{3}(1 + V^\pi(s_1)) + \frac{2}{3}(-1 + V^\pi(s_2))\right) + \\ + \frac{1}{2}\left(\frac{1}{3}(-1 + V^\pi(s_3)) + \frac{2}{3}(1 + V^\pi(s_4))\right).$$

Рахуємо далі. Розглянемо для прикладу стан s_1:

$$V^\pi(s_1) = \frac{1}{2}\left(\frac{1}{2}(-5 + V^\pi(s_5)) + \frac{1}{2}(5 + V^\pi(s_6))\right) + \\ + \frac{1}{2}\left(\frac{1}{2}(5 + V^\pi(s_5)) + \frac{1}{2}(-5 + V^\pi(s_6))\right).$$

Оскільки після другого кола гра закінчується, $V^\pi(s_5) = V^\pi(s_6) = 0$, а значить, $V^\pi(s_1) = 0$ теж. Легко бачити, що для інших станів це теж вірно: $V^\pi(s_2) = V^\pi(s_3) = V^\pi(s_4) = 0$, а значить, і $V^\pi(s_0) = 0$.

Отже, підкидаючи монетку на кожному колі, ми нічого не виграємо і нічого не програємо. Оптимальна це стратегія або можна все-таки щось виграти? Давайте спробуємо відповісти на це питання.

Але давайте спочатку для простоти припустимо, що ми вже точно знаємо нашу модель. Завдання — знайти оптимальну стратегію поведінки для

агента в цій моделі. У такій постановці ми вже вміємо шукати $V^{\pi}(s)$, вирішуючи рівняння Беллмана, але різних стратегій ще більше, ніж станів, і перебрати їх ми зазвичай не в змозі.

На щастя, це і не обов'язково: нам потрібно тільки навчитися підраховувати оптимальне значення стану, тобто шукати очікуваний сумарну прибуток, який отримає агент, якщо почне з цього стану і буде слідувати оптимальної стратегії:

$$V^*(s) = \max_{\pi} \mathbb{E} \sum_{t=0}^{\infty} \gamma^t r_t.$$

Цю функцію можна визначити як розв'язок рівнянь:

$$V^*(s) = \max_a \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma V^*(s'))$$

а потім обрати необхідну стратегію, виходячи з підрахованих значень V^*:

$$\pi^*(s) = \arg\max_a \sum_{s' \in S} P^a_{ss'}(R^a_{ss'} + \gamma V^*(s')).$$

Як вирішувати рівняння? Вони вже не лінійні, і вирішити їх точно і ефективно не вийде, але наша справа все одно аж ніяк не безнадійна. Як відомо, якщо складне рівняння представлено у вигляді $x = f(x)$, то його можна вирішувати ітеративно методом Ньютона: почати з якогось x_0 і послідовно перераховувати $x_{k+1} = f(x_k)$, поки процес не зійдеться, тобто поки зміни $|x_{k+1} - x_k|$ не стануть зовсім маленькими. Тут ця ідея чудово працює, адже, як легко помітити, рівняння вже представлені в потрібному вигляді!

Це можна робити і для вихідних лінійних рівнянь, вийде швидше, ніж вирішувати систему по-чесному. Природно, в результаті виходить наближений, чисельний метод рішення рівнянь Беллмана, але в машинному навчанні нам нічого іншого і не потрібно.

Так що, щоб підрахувати функції значень станів для даної стратегії π, можна просто ітеративно перераховувати їх по рівняннях Беллмана:

$$V^{\pi}(s) := \sum_{a} \pi(s,a) \sum_{s' \in S} P^{a}_{ss'}(R^{a}_{ss'} + \gamma V^{\pi}(s')),$$

поки процес не зійдеться.

А для оптимальних значень ми будемо перераховувати рівняння з максимумами замість математичних сподівань:

$$V^{*}(s) := \max_{a} \sum_{s' \in S} P^{a}_{ss'}(R^{a}_{ss'} + \gamma V^{*}(s')).$$

Рівно те саме можна зробити для функції $Q(s,a)$:

$$Q(s,a) := \sum_{s' \in S} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma \sum_{a'} \pi(s,a') Q(s,a') \right),$$

до збігу. І з оптимальним $Q^{*}(s,a)$ все так само:

$$Q^{*}(s,a) := \sum_{s' \in S} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma \max_{a'} Q^{*}(s,a') \right),$$

поки не зійдеться; потім можна обчислити оптимальну функцію значень як $V^{*}(s) := \max_{a} Q^{*}(s,a)$.

Підрахуємо для прикладу значення $V^{*}(s)$ і $Q^{*}(s,a)$ з прикладу з рис. 1.2:

$$V^{\pi}(s_0) = \max \left\{ \frac{1}{3}(1 + V^{\pi}(s_1)) + \frac{2}{3}(-1 + V^{\pi}(s_2)) \,, \right.$$
$$\left. \frac{1}{3}(-1 + V^{\pi}(s_3)) + \frac{2}{3}(1 + V^{\pi}(s_4)) \right\}.$$

Гра все ще закінчується на s_5 та s_6, тому $V^*(s_5) = V^*(s_6) = 0$. Скористаємось цим, щоб порахувати V^* після першого кола гри:

$$V^*(s_1) = \max\left\{\frac{1}{2}(-5) + \frac{1}{2}(5), \frac{1}{2}(5) + \frac{1}{2}(-5)\right\} = 0$$

Аналогічно і $V^*(s_4) = 0$. А от для $V^*(s_2)$ і $V^*(s_3)$ ситуація більш оптимістична:

$$V^*(s_2) = \max\{-5, 5\} = 5, V^*(s_3) = \max\{-5, 5\} = 5.$$

Підставляючи це у вираз для $V^*(s_0)$, отримаємо:

$$V^*(s_0) = \max\{\frac{1}{3} + \frac{2}{3}(-1 + 5), \frac{1}{3}(-1 + 5) + \frac{2}{3}\} = 3.$$

Ми можемо виграти, причому з чималим в середньому рахунком! Щоб дізнатися, як виграти, потрібно підрахувати функцію Q^*; зробимо це для стану s_0, підставляючи інші значення відразу (вони рахуються очевидним чином):

$$\begin{aligned} Q^*(s_0, even) &= \frac{1}{3}(1 + \max_a Q^*(s_1, a)) + \frac{2}{3}(-1 + \max_a Q^*(s_2, a)) = \\ &= \frac{1}{3}(1 + 0) + \frac{2}{3}(-1 + 5) = 3, \\ Q^*(s_0, odd) &= \frac{1}{3}(-1 + \max_a Q^*(s_3, a)) + \frac{2}{3}(1 + \max_a Q^*(s_4, a)) = \\ &= \frac{1}{3}(-1 + 5) + \frac{2}{3}(1 + 0) = 2. \end{aligned}$$

Це означає, що на першому колі потрібно вибирати непарне число, що не має великої вірогідності перемоги (ми ж знаємо, що противник любить непарні числа), а навпаки: потрібно спочатку піддатися противнику, щоб заспокоїти його і змусити повторити свою дію — тоді-то ми його і дістанемо,

а друге коло куди важливіше першого. І весь цей хитрий план вийшов природним чином з рівнянь Беллмана.

На рис. 1.3 зображено деякі результати наших обчислень: сірим на ньому показані ті стану і дії, в які ми ніколи не потрапимо, якщо будемо слідувати оптимальної стратегії, що задається Q^*, а чорним — ті, в які потрапити можемо (в стані s_1 було все одно, яку дію вибирати, так що вибрали парне число).

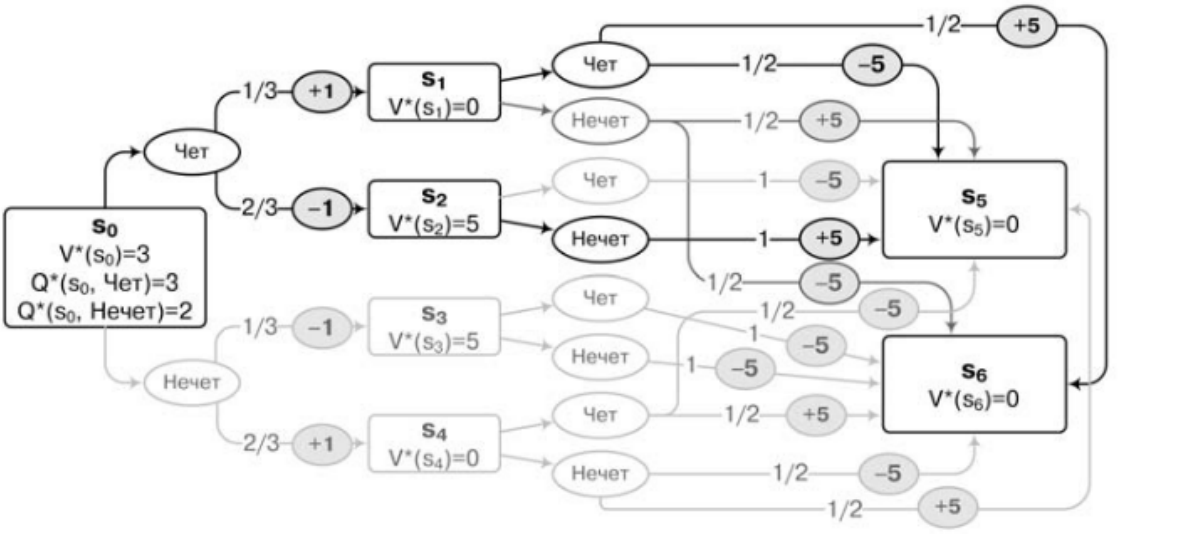

Рисунок 1.3 — Гра пара-непара: значення функцій стану та оптимальна стратегія

Зауважимо, що перерахунок в нашому алгоритмі використовує інформацію від усіх можливих станів-попередників; оскільки станів зазвичай дуже багато, всі ці формули поки що фактично не застосовуються на практиці. Але можна сформулювати таку саму процедуру і для одного тренувального прикладу, який складається з поточного стану s,виконаної дії a, стану s', в яке ми після цього перейшли, і безпосередньої нагороди r:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right).$$

Теоретичні гарантії у цього метода з'являються, якщо кожна пара (s,a)

зустрічається в процесі навчання нескінченну кількість разів, s' обирають з розподілу $P^a_{ss'}$, а r генерується з середнім $R(s,a)$ і обмеженою дисперсією.

Втім, насправді наша мета — не функції V і Q, а оптимальна стратегія π^*, і шукаємо ми V^π тільки потім, щоб поліпшити π. Як це можна зробити?

Отже, ми досить довго вже міркували про функції значень станів і пар стан-дія, багато зрозуміли, а тепер пора переходити до найцікавішого. Практично всі сучасні підходи до навчання засновані на дуже простому, але дуже потужному принципі, який називається TD-навчання (TD-learning), від слів temporal difference (тимчасові різниці). Загальний принцип TD-навчання такий: давайте навчати стан на основі вже навчених нами оцінок для наступних станів. Кожен раз, коли ми робимо черговий перехід, ми трошки підтягуємо функцію V для того стану (або функцію Q для пари стан-дія), з якого ми вийшли, до значення функції V для того стану (або функції Q для пари стан-дія), в який ми потрапили.

Найпростіший алгоритм TD-навчання, так зване TD(0)-навчання, виглядає так. Спочатку потрібно ініціалізувати функцію $V(s)$ і стратегію π випадковим чином, а потім на кожному епізоді навчання:

- ініціалізувати s;
- для кожного кроку в епізоді:
  - обрати a по стратегії π;
  - виконати a, поспостерігати результат r та наступний стан s;
  - оновити функцію V в стані s по формулі:

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s));$$

  - перейти до наступного кроку, при цьому $s := s'$.

От і все! На перший погляд здається, що тут відбувається якась чорна магія: ми навчаємо $V(s)$ на основі інших значень $V(s')$ але їх ми теж ініціалізували випадковим чином! Однак все працює: сенс в тому, щоб використовувати вже навчені закономірності для пошуку більш глибоких закономірностей. Спочатку навчаться значення $V(s)$ на станах, які безпосередньо ведуть до справжніх нагород r, а потім ці значення будуть поступово передавати

накопичені в них знання далі, до попередніх станів. В результаті навчання вийде цілеспрямованим, TD-навчання набагато швидше і ефективніше, ніж інші стратегії.

Втім, і його можна поліпшити. TD(0) дивиться на один крок вперед; можна, можливо розглянути алгоритм, який буде оновлювати стану відразу на багато кроків назад. Він називається TD(λ), і λ тут грає ту ж роль, що і раніше: ми оновлюємо кожен стан u за формулою:

$$V(u) := V(u) + \alpha(r + \gamma V(s') - V(s))\varepsilon(u)$$

на основі значення $\varepsilon(u)$, яке показує, наскільки часто цей стан відвідували в минулому. Значення $\varepsilon(u)$ точно так же експоненційно загасають з показником λ:

$$\varepsilon(s) = \sum_{k=1}^{t} \lambda^{t-k}[s = s_k],$$

де $[s = s_k]$ дорівнює одиниці, якщо $s = s_k$, і нулю в інших випадках.

Якщо $\lambda = 0$, TD(λ) перетворюється в TD(0). А значення $\varepsilon(u)$ можна також зберігати і перераховувати в реальному часі після кожного нового переходу:

$$\varepsilon(u) := \begin{cases} \lambda\varepsilon(u) + 1, & \text{якщо поточний стан — це u,} \\ \lambda\varepsilon(u) & \text{в усіх інших випадках.} \end{cases}$$

Звичайно, на практиці оновлюють не всі стани, а кілька з найбільшими значеннями $\varepsilon(u)$, які в реальному реалізації зазвичай зберігаються в пріоритетній черзі.

На даний алгоритм принцип TD-навчання можна перетворити різними способами. Якщо реалізувати його для функції Q зовсім в лоб, вийде алгоритм SARSA, який представляє собою on-policy TD-навчання, після кожного чергового переходу $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ робить наступне оновлення:

$$Q(S_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)).$$

Аналогічно, SARSA(λ) оновлює усі значення на усіх парах (s, a)з урахуванням $\varepsilon(s, a)$:

$$Q(s, a) := Q(s, a) + \alpha[r_{t+1} + \gamma Q(s_{t+a}, a_{t+1}) - Q(s_t, a_t)]\varepsilon(s, a),$$
$$\varepsilon(s, a) := \gamma\lambda\varepsilon_{t-1}(s, a) + [s = s_t, a = a_t].$$

При цьому, правда, стратегія повинна бути м'якою, наприклад ε-жадібної з ε, що зменшується , щоб алгоритм міг досліджувати нові можливі дії, але з часом вона повинна якось плавно сходитися; це вносить додаткові інженерні складності в реалізацію, тому що від вибору характеру загасання ε або другого параметра нежадібності може багато чого залежати.

Тому більш популярно off-policy TD-навчання функції Q, яке зазвичай так і називається Q-навчанням [10]. Тут ми відразу вирішуємо рівняння Белльмана щодо максимумів:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right).$$

Тепер Q безпосередньо наближає оптимальну функцію Q^*, незалежно від стратегії; це означає, що ми можемо дотримуватися абсолютно будь-якої стратегії, а навчатися все одно будуть правильні оптимальні значення Q^*. Аналогічно можна визначити і $Q(\lambda)$:

$$Q(s, a) := Q(s, a) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]\varepsilon_t(s, a),$$
$$\varepsilon_t(s, a) := \gamma\lambda\varepsilon_{t-1}(s, a) + [s = s_t, a = a_t],$$

тільки тепер ще й потрібно забувати сліди, якщо ми не слідуємо стратегії: $\varepsilon_t(s,a) := 0$, якщо $Q(s_t,a_t) \neq \max_a Q(s_t,a)$.

1.3 DQN

Всі ті алгоритми навчання з підкріпленням, про які ми до цих пір говорили, використовували значення виду $Q(s,a)$ або $V(s)$, причому вони намагалися отримати ці значення в явному вигляді, наприклад, навчити функцію Q^* на всіх можливих входах (s,a), щоб потім максимізувати очікуваний результат, знайшовши оптимальну стратегію. Але ж цих самих станів s зазвичай астрономічне число — уявіть, скільки можливих позицій в грі го! А якщо число станів помножити на число можливих дій в них, вийде ще більше. Тому в реальності, звичайно, ніхто не намагається перерахувати всі можливі стани, побудувати і навчити величезну таблицю $Q(s,a)$. Підхід зазвичай такий:

- давайте уявимо входи, тобто стани $s \in S$ і дії $a \in A$, у вигляді якихось характерних ознак, так, щоб розмірність входу перестала бути астрономічною;
- а функцію $Q(s,a)$, в якій раніше значення на різних входах були незалежними один від одного, уявімо як якусь параметричну модель машинного навчання $Q(s,a;\theta)$, на вхід якої подаються ознаки, що описують s і a;
- тоді функція $Q(s,a;\theta)$ — це просто складна функція з ознак в одне дійсне число (очікуваний виграш, або його ймовірність у випадку бінарного результату), і її параметри θ можна намагатися навчати методами ма- шинного навчання;
- входами для навчання буде, згідно з ідеєю TD-навчання, кожен черговий перехід $(s_t, a_t, r_{t+1}, s_{t+1})$;
- і кожен крок навчання виглядає так: агент робить хід a зі стану s, переходить в новий стан s', отримуючи за це безпосередню нагороду r, а потім робить один крок навчання системи $Q(s,a;\theta)$ зі входом (s,a) і виходом $\max_{a'} Q(s',a';\theta) + r$ (нагадаємо, що в переважній більшості

випадків $r = 0$, нагороду зазвичай дають тільки в самому кінці епізоду навчання); окрім того, агент може також зробити такі кроки по відношенню до попередніх позицій, оновивши ваги не тільки для останнього входу, але і для декількох попередніх.

У цій схемі нейронні мережі, як правило, добре працюють в якості універсального чорного ящика, який може наблизити будь-яку функцію, в тому числі і функцію $Q(s, a)$. Наприклад, якщо під навчанням розуміти звичайний градієнтний спуск, то ми оновлюємо ваги нейронної мережі за таким правилом:

$$w_{t+1} = w_t + \alpha(y_{t+1} - y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w y_k,$$

де y_k — вихід нейронної мережі ан кроці k, а λ — це параметр затухання, який визначає в якій мірі необхідно враховувати попередні кроки. Виходить, що при $\lambda = 0$ ми оновлюємо ваги тільки на основі останнього ходу, а при $\lambda = 1$ розглядаємо усю історію від самого початку.

Перші успіхи нейронних мереж у ролі навчання з підкріпленням відносяться до часів досить давніх. У 1992 році Джеральд Тезауро (Gerald Tesauro) розробив програму, що отримала назву TD-Gammon [11, 12]; назва цілком логічна, адже грала ця програма в нарди (backgammon), і робила це за допомогою TD-навчання. TD-Gammon працює в точності як написано вище, використовуючи звичайну неглибоку нейронну мережу з одним прихованим шаром для навчання за правилом вище. Нарди виявилися дуже благодатним грунтом для такого підходу бо хід гри залежить від кидків кубика, а це значить, що можна досліджувати значну частину простору пошуку, просто граючи з самим собою, кубики подбають про усі необхідні випадковості. І це саме те, що робила TD-Gammon для навчання: просто грала сама з собою мільйони партій, поступово навчаючись все краще і краще; чари тут в тому, що ніякого навчального набору виявляється не потрібно.

TD-Gammon чекав успіх. З точки зору навчання добрим знаком було те, що модель добре масштабувати: при збільшенні розміру мережі і часу,

виділеному на навчання, мережа починала грати все краще. У міру навчання TD-Gammon спочатку навчалася найпростішим елементам стратегії і тактики нард, а потім поступово починала виділяти складніші ознаки. В результаті програма навіть на простому поданні позиції без всяких хитростей починала грати дуже сильно, а при додаванні до подання кількох вручну породжених ознак з більш ранньої програми Neurogammon TD-Gammon починала успішно змагатися з людьми-чемпіонами.

Єдиними слабкостями TD-Gammon були помилки у використанні подвоєння ставок і погана гра в ендшпілі: TD-Gammon дивилася тільки на два ходи вперед, а ендшпілі навіть в нарди вимагають більш глибокого розрахунку. Тому в виставочних іграх TD-Gammon трохи поступилася тодішнім чемпіонам. Тим не менше, TD-Gammon зробила серйозний вплив на розвиток нард: деякі класичні позиції були повністю переоцінені гравцями-людьми, тому що оцінка TD-Gammon суперечила людської інтуїції і практиці ігор; той же ефект можна було спостерігати пізніше і в шахах, а тепер і в го.

Однак TD-Gammon надовго залишилася практично єдиною успішною програмою, заснованої на ідеях навчання нейронної мережі з підкріпленням. Дослідники тут же спробували застосувати аналогічних підхід до шахів і го, але у них мало що вийшло. Далі були навіть песимістичні роботи, які показували, що Q-навчання з нелінійними параметричними наближеннями (а нейронна мережа — це саме нелінійне наближення, яке тільки буває) часто розходиться, а успіх TD-Gammon пояснювався виключно згаданим вище ефектом рівномірного розподілу навчання по простору пошуку за рахунок кубиків.

На щастя, ці песимістичні прогнози не виправдалися; у міру того як розвивалася революція глибокого навчання, почали з'являтися і спроби моделювати функції V і Q, а також навколишнє агента середу, за допомогою глибоких мереж. Після ранніх робіт прорив, остаточно визначив направлення для сучасних успіхів, був досягнутий в роботі Володимира Мніха (Volodymyr Mnih) зі співавторами з компанії Google DeepMind, в якій вони застосували ідеї навчання з підкріпленням до ранніх, але від цього не менш привабливих ігор для приставок і аркадних автоматів Atari.

Перша робота була викладена на arXiv в 2013 році [13], а трохи поліпшена і застосована до більшої кількості різних ігор модель була описана в статті 2015 року, що вийшла в одному з головних наукових журналів світу, Nature [14]. Цей підхід отримав назву глибокого навчання з підкріпленням (deep reinforcement learning), а мережі, навчені таким способом, називаються глибокими Q-мережами (Deep Q-networks, DQN).

У DQN в варіанті [14] є деякі невеликі, але важливі удосконалення по відношенню до базової архітектурі, описаної вище. По-перше, практика показує, що навчатися безпосередньо на послідовних кадрах гри — погана ідея: сусідні кадри занадто схожі один на одного, сильно корелюють, причому з часом їх розподіл, природно, зсувається в залежності від ходу гри, але залишається локалізованим.

Це заважає ефективному навчанню, адже у звичайній постановці завдання навчання ми припускаємо, що тренувальні дані незалежні, а розподіл даних з часом не змінюється. Тому в міру навчання DQN спочатку накопичує деякий досвід, зберігаючи свої дії і їх результати протягом якогось часу, а потім вибирає з цього досвіду випадковий міні-батч окремих прикладів для навчання, взятих у випадковому порядку; для накопичення досвіду при навчанні використовувалася ε-жадібна стратегія. Формально кажучи, на кожному кроці навчання t ми:

- вибираємо наступне дію a_t (в ε-жадібної стратегії ми вибираємо випадкову дію з ймовірністю ε і $a_t = \arg\max Q(s_t, a; \theta)$ в іншому випадку;
- робимо цю дію, отримуючи нагороду r_t і наступний стан s_{t+1}; нову одиниця досвіду (s_t, a_t, r_t, s_{t+1}) записується в пам'ять;
- потім вибираємо з пам'яті випадковий міні-батч таких одиниць досвіду для навчання; для простоти нехай це буде одна одиниця (s_j, a_j, r_j, s_{j+1});
- підраховуємо вихід мережі y_j (про це трохи нижче) і робимо один крок градієнтного спуску для функції помилки $L = (y_j - Q(s_j, a_j; \theta))^2$; це означає що ми зрушуємо ваги мережі на

$$\nabla_\theta L = 2(y_j - Q(s_j, a_j; \theta))\nabla\theta Q(s, a; \theta).$$

По-друге, важливу роль для успіху зіграло те, що при навчанні DQN мережу, яка відповідала за цільову функцію (target network), була відділена від мережі, яка власне навчається. Практика показує, що якщо застосовувати TD-навчання безпосередньо, занадто потужні апроксиматори (наприклад, нейронні мережі) виявляють безсумнівні схильності до галюцинацій: вони швидко заходять в якісь ними самими придумані локальні екстремуми і починають дуже глибоко досліджувати абсолютно безглузді частини простору пошуку, безцільно витрачаючи ресурси і практично не навчаючись.

На щастя, це відносно легко виправити: досить зробити так, щоб мережа не відразу використала оновлену версію в цільової функції, а навчалася досить довгий час за старими зразками, перш ніж зробити вже повноцінний глобальний апдейт. Іншими словами, в наведеному вище алгоритмі ми рахуємо

$$y_j = \begin{cases} r_j, & \text{якщо епізод навчання закінчився,} \\ r_j + \gamma \max_{a'} Q(s_j, a_j; \theta_0) & \text{якщо ні,} \end{cases}$$

де θ_0 — це такі собі зафіксовані ваги, що не змінюються після кожного тестового прикладу, змінюються тільки θ. В якийсь момент, зазвичай один раз за один або кілька епізодів навчання, потрібно повертатися до цих вагів цільової функції і привласнювати $\theta_0 := \theta$.

Де-факто у нас паралельно навчаються дві мережі: одна визначає поведінку, а інша — цільову функцію; структура у них одна і та ж, і навчаються вони однаково на одних і тих же даних, але одна мережа поступово відстає від іншої, час від часу стрибком наздоганяючи її.

По-третє, варто відзначити, що на практиці зазвичай застосовується архітектура, в якій потенційний вплив агента $a \in A$ не подається на вхід, а просто у мережі стільки виходів, скільки можливих дій, і на вході $s \in S$ мережа намагається передбачити результати кожної дії (після чого, природно, вибирає максимальне). Це важливе поліпшення, тому що воно дозволяє отримати відповіді для відразу всіх дій за один прохід по мережі, що прискорює

в рази, а мережа від цього сильно складніше не стає, адже основна частина її логіки залишається колишньою і використовується заново.

І нарешті, ще одне важливе поліпшення полягає в переході до так званої dueling network: ми поділяємо Q-мережу на два канали, один з яких обчислює оцінку позиції $V(s;\theta)$, яка є функцією позиції і не залежить від теперішньої дії a, а інший — залежить від дії advantage function (функції переваги) $A(s,a;\theta)$. На останньому етапі вони просто складаються:

$$Q(s,a;\theta) = V(s;\theta) + A(s,a;\theta).$$

Таким чином, одна частина мережі навчається оцінювати позицію як таку, а інша — передбачати, наскільки корисні будуть різні наші дії в цій позиції. До речі, зверніть увагу, що ми, звичайно ж, не можемо відрізнити одне від іншого в навчальній вибірці, це всього лише злегка змінена архітектура мережі, а навчаємо ми як і раніше функцію $Q(s,a;\theta)$, але ця зміна в архітектурі дає потрібний натяк і часто істотно покращує результати.

У роботах [13, 14] цей підхід застосували до того, щоб грати в ігри Atari, причому входом служив не стан гри (це вимагало б окремих інженерних рішень для кожної гри, а саме цього і хотілося б уникнути), а власне ігрове поле у вигляді картинки з пікселів, тієї самої, яку бачить на екрані гравець. Точніше кажучи, на вхід подавалася конкатенація останніх чотирьох кадрів гри, тобто можна сказати, що довгої пам'яттю агента не забезпечили, він ледь-ледь міг оцінити швидкості різних об'єктів на екрані.

Єдине послаблення, яке зробили агенту, полягало в тому, що його не змушували вчити читати цифри очок, а просто давали число очок з гри в явному вигляді (замазуючи його при цьому на картинках); в результаті прийшли навіть до ще більш простої схеми: видавали нагороду +1 за кожне позитивне досягнення в грі і -1 за кожну негативну подію (в Atari це зазвичай втрата чергового життя).

В результаті вийшло, що така мережа, нічого не знаючи власне про правила гри, просто по вхідному зображенню і цільовій функції навчилася грати в багато ігор Atari краще людини. Цікаво, однак, що не в усі. Ігри без дов-

гострокової стратегії на кшталт Breakout або Video Pinball підкорилися DQN без проблем, результати були вдесятеро вище людських (нагадаємо, що мова йде про людей-експертів, кращих гравців світу у відповідній грі). Результати на рівні людських або трохи краще вийшли в іграх на кшталт Pong і Space Invaders, де стратегія є, але вона не дуже обов'язкова і не дуже довгострокова. А ось в іграх, де потрібно думати надовго вперед, нічого не вийшло; наприклад, в Montezuma's Revenge DQN грати абсолютно не навчилася, стійко отримуючи нульові результати.

І ще одне зауваження: навчання з підкріпленням для ігор та інших подібних відрізняється від більшості інших завдань машинного навчання тим, що тут у нас фактично необмежене джерело нових тренувальних прикладів. У випадку ігор Atari ми можемо запускати симулятор гри скільки завгодно раз, пробуючи різні стратегії і навчаючи модель добре грати. У разі гри в нарди або го модель може скільки завгодно грати сама з собою, теж отримуючи практично необмежену послідовність прикладів для навчання. І хоча приклади ці будуть в певному сенсі залежати від поточної версії моделі - в кінці кінців, саме вона (зазвичай з доданим випадковим шумом для дослідження) грає, коли створюються нові тренувальні приклади, — це все одно не йде ні в яке порівняння зі звичайними ситуаціями, коли є якийсь фіксований датасет, і максимум, що ви можете зробити, — додати в нього трохи випадкового шуму. З іншого боку, навчання оптимальної стратегії методами навчання з підкріпленням в будь-який досить складній ситуації — це процес довгий і складний: ті самі моделі DQN для ігор Atari навіть на найсучасніших відеокартах навчаються за кілька днів, перш ніж можуть показати якісь розумні результати

Тому цілком логічно, що в глибокому навчанні з підкріпленням основні сили подальших досліджень поки що зосереджені не на максимально ефективному використанні кожного тренувального прикладу (їх легко нагенерувати ще), а на тому, щоб максимально швидко навчатися.

Наступний прорив у швидкості навчання був зроблений на основі асинхронного навчання з підкріпленням, що використовує дві особливості навчання DQN:

- по-перше, навчання відбувається не після кожного ходу, а шляхом випадкового вибору з накопиченої пам'яті, і для навчання потрібно спочатку зібрати деяку кількість тестових прикладів, а тільки потім оновлювати ваги моделі;
- по-друге, мережа, яка генерує цільові значення для функції втрат, — це не та ж сама мережа, яка навчається після кожного міні-батчу, вона може і навіть повинна суттєво відставати від мережі, яка генерує поведінку агента.

Тому виявилося, що навчання з підкріпленням можна розбити на кілька практично незалежних частин, які повинні ділитися один з одним новинами тільки в певні досить далеко один від одного віддалені моменти часу, а між ними можуть працювати абсолютно паралельно і незалежно:

- повинен все-таки бути якийсь центральний процес, сервер, який зберігає поточні значення параметрів, оновлює їх у міру потреби і роздає всім іншим;
- перший вид процесів — це власне гравці, які взаємодіють з навколишнім світом і напрацьовують новий досвід; їм потрібно час від часу отримувати від сервера оновлення параметрів моделі (вони використовуються при виборі дій), а самі вони просто накопичують одиниці досвіду в вигляді тих самих четвірок (s_t, a_t, r_t, s_{t+1}) і передають накопичений досвід в загальне сховище пам'яті;
- другий вид процесів, вчителі, отримують зі сховища пам'яті досвід у вигляді міні-батчів одиниць досвіду і вважають градієнти функції помилки; їм потрібна для цього мережа, яка генерує цільові значення, і поточна, так що вчителі знаходяться в більш тісному контакті з сервером; але зауважимо, що вони як і раніше абсолютно незалежні, кожен з них вважає своє власне значення градієнта і свої власні поновлення для вагів моделі;
- нарешті, власне сервер збирає всі ці оновлення, застосовує їх до збереженої у нього моделі, і в якийсь момент (зазвичай регулярний, але достатній точно рідкісний) роздає оновлену модель назад гравцям і вчителям, а також оновлює модель, яка генерує цільові значення; виходить,

що синхронізація в такій архітектурі, звичайно, потрібна, але її можна робити відносно рідко.

В роботі [15] цей підхід був застосований для того, щоб розпаралелити навчання з підкріпленням на кластер з декількох комп'ютерів: в розробленій авторами архітектурі з характерною назвою Gorila (від слів General Reinforcement Learning Architecture) кожен з процесів, описаних вище, може бути реалізований на окремому комп'ютері.

А потім з'ясувалося, що можна розпаралелити все це ще ефективніше, якщо в якості окремих агентів використовувати потоки одного і того ж процесора; тоді вони можуть просто звертатися до однієї і тієї ж моделі в пам'яті, але при цьому все одно робити свою справу відносно незалежно.

Ми зобразили архітектуру Gorila на рис. 1.4. Зверніть увагу, що розпаралелено тут буквально все. По-перше, кілька незалежних гравців (Actors) діють кожен у своїй копії середовища, породжуючи нові одиниці досвіду й передаючи їх в глобальну пам'ять (replay memory). По-друге, учителів (learners) теж кілька: кожен містить копію Q-мережі і обчислює градієнти за своїм чергового міні-батчем тренувальних прикладів, взятому з пам'яті; для підрахунку градієнтів використовується цільова Q-мережу (target network). По-третє, градієнти ці відправляються на сервер параметрів, який теж тримає кілька паралельних вузлів (shards), кожен з яких зберігає і оновлює свою окрему частину вектора параметрів мережі θ.

Цікаво, що асинхронне навчання виявляється не просто швидше, але і суттєво краще. Причини такого ефекту точно не відомі, але, схоже, справа в тому ж ефекті, який допомагає стохастичному градієнтному спуску: розподілене асинхронне навчання призводить до того, що модель не намагається занадто детально дослідити одну і ту ж частину простору пошуку. Раз гравці поводяться по-різному, і їх упереджені ставлення до різних частин простору пошуку усереднюються. В роботі [16] перші результати DQN на іграх Atari були перевищені в рази при тому, що загальний час навчання зменшився; цікаво, що для навчання при цьому не використовувалися відеокарти, і все робилося в 16 потоків на звичайному сучасному процесорі.

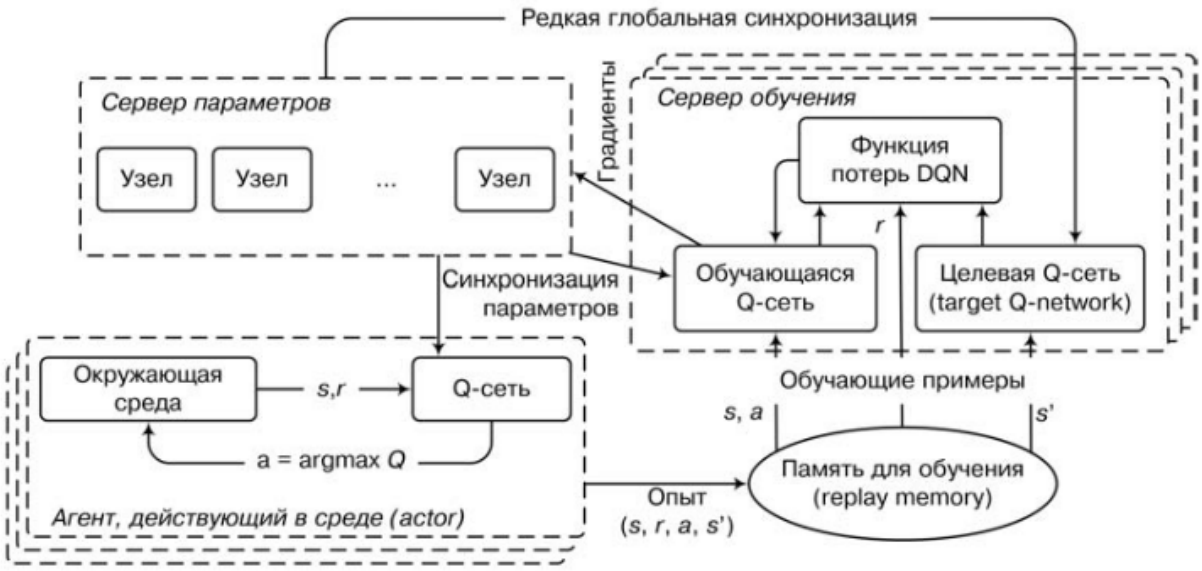

Рисунок 1.4 — Архітектура Gorila

Висновки до розділу 1

В даному розділі проведено аналіз навчання з підкріпленням. В межах цього аналізу було досліджено задачу навчання з підкріпленням у найпростішому випадку — multi-armed bandits, в якому стан системи є незмінним. Далі на основі цього простого випадку побудовано загальний фреймворк для розв’язку задач навчання з підкріпленням у випадку обмеженої кількості станів та дій. Далі, запропоновано метод Q-навчання у випадках коли розмірність станів та дій настільки велика, що ніякої можливості перебрати їх усіх принципово не можливо, а також запропоновано підхід до швидкого навчання таких мереж.

РОЗДІЛ 2

НАВЧАННЯ З ПІДКРІПЛЕННЯМ ТА ФІНАНСОВІ РИНКИ

Науково-дослідна спільнота Deep Learning значною мірою трималася осторонь фінансових ринків. Можливо, це тому, що фінансова галузь має погану репутацію, проблема не здається цікавою з точки зору дослідження, або тому, що дані важко і дорого отримати. У цьому розділі буде показано, що навчання агентів з підкріпленням для торгівлі на фінансових ринках може бути цікавою дослідницькою проблемою.

2.1 Основи інфраструктури фінансового ринку

Торгівля на криптовалютних (і більшості фінансових) ринків відбувається на так званих безперервних подвійних аукціонах з книжкою відкритих замовлень на біржі. Це означає, що є покупці та продавці, яких біржа зводить так, щоб вони могли торгувати один з одним. Існує кілька десятків бірж, і кожна може торгувати дещо різними продуктами (наприклад, Bitcoin або Ethereum за американський долар). З точки зору інтерфейсу, і щодо даних, які вони надають, вони виглядають приблизно однаково.

На рис. 2.1 показано інтерфейс біржи GDAX — однієї з найпопулярніших бірж в США.

Розглянемо по порядку зображені на рисунку елементи:

- Графік цін (посередині). Поточна ціна — це ціна останньої торгівлі. Вона змінюється залежно від того, чи була ця торгівля покупкою чи продажем (докладніше про це нижче). Графік цін зазвичай відображається у вигляді японських свічок, який показує ціни відкритого / стартового (O), високого (H), низького (L) та закриття / закінчення (C) у визначеному часовому вікні. На рис. 2.1 вище цей період становить 5 хвилин, але ви можете змінити його, використовуючи спадне меню. У стовпчиках нижче діаграми цін відображають об’єм (V), який є загальним обсягом усіх торгів, що відбулися в той період. Обсяг важливий, оскільки він

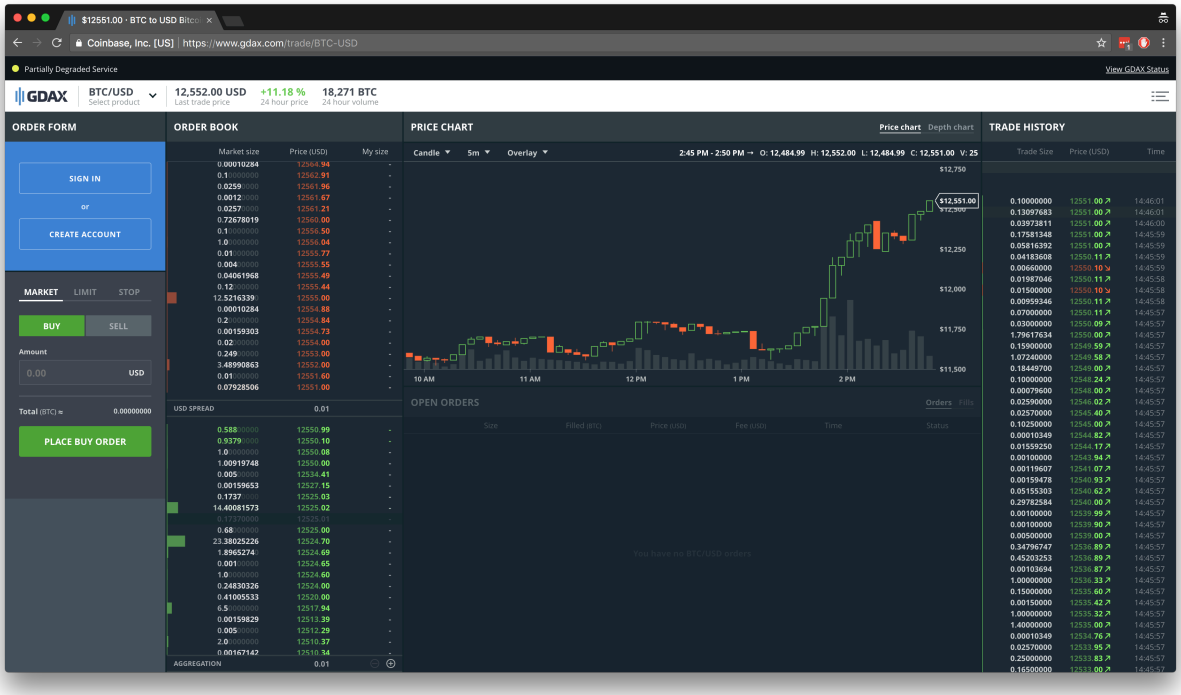

Рисунок 2.1 — Інтерфейс біржи DGAX

дає відчуття ліквідності ринку. Якщо ви хочете купити біткоін на 100 000 доларів, але продати ніхто не бажає, ринок неліквідний. Ви просто не можете купити. Високий обсяг торгівлі вказує на те, що багато людей готові до угод, а це означає, що ви, ймовірно, зможете придбати чи продати, коли захочете це зробити. Взагалі кажучи, чим більше грошей ви хочете вкласти, тим більше обсяг торгівлі ви хочете. Обсяг також вказує на якість тенденції цін. Високий об'єм означає, що ви можете розраховувати на рух цін більше, ніж якби був низький обсяг. Високий обсяг часто (але не завжди, як у випадку з ринковими маніпуляціями) є консенсусом великої кількості учасників ринку.

- Історія торгів. Права сторона показує історію всіх останніх торгів. Кожна торгівля має розмір, ціну, часову позначку та напрямок (купівля чи продаж). Торгівля — це договір між двома сторонами, що приймають та виробляють. Детальніше про це нижче.
- Книга замовлень (зліва). Ліва сторона показує книгу замовлень, яка містить інформацію про те, хто готовий купувати та продавати за якою

ціною. Книга замовлень складається з двох сторін: Запити (також називаються пропозиції) та Ставки. Запити — це люди, які бажають продати, а торги — люди, які бажають купувати. За визначенням, найкращий запит, — найнижча ціна, за яку хтось готовий продати, більша, ніж найкраща пропозиція, — найвища ціна, за яку хтось готовий купити. Якби цього не було, торгівля між цими двома сторонами вже відбулася б. Різниця між найкращим запитом та найкращою ставкою називається спред. Кожен рівень книги замовлень має ціну та обсяг. Наприклад, обсяг 2,0 при рівні ціни 10 000 доларів означає, що ви можете придбати 2 BTC за 10 000 доларів. Якщо ви хочете придбати більше, вам доведеться заплатити більш високу ціну за суму, що перевищує 2 BTC. Обсяг на кожному рівні сукупний, це означає, що ви не знаєте, скільки людей або замовлень складається з 2 BTC. Можливо, одна людина продає 2 BTC, або може бути 100 людей, що продають 0,02 BTC кожен (деякі біржі надають цей рівень інформації, а більшість не надають). Приклад на рис. 2.2.

То що відбувається, коли ви відправляєте замовлення на придбання 3 BTC? Ви купуєте (округлюючи) 0,08 BTC за $12,551,00, 0,01 BTC за $12,551,6 і 2,91 BTC за $12,552,00. У GDAX ви також платите 0,3% комісійної плати, загалом близько 1,003 * (0,08 * 12551 + 0,01 * 12551,6 + 2,91 * 12552) = $37,768,88 та середня ціна за BTC 37768,88 / 3 = $12,589,62. Важливо зауважити, що те, що ви насправді платите, набагато вище, ніж $12 551,00, яка була поточна ціна! 0,3% комісія на GDAX надзвичайно висока порівняно з комісіями на фінансових ринках, а також набагато вища, ніж комісія багатьох інших бірж криптовалют, які часто становлять від 0% до 0,1%.

Також зауважте, що вашим замовленням на купівлю було витрачено весь обсяг, який був доступний на рівні $12,551.00 та $12,551,60. Таким чином, книга замовлень рухатиметься вгору, і найкращим запитом стане $12 552,00. Поточна ціна також стане 12 552,00 доларів, адже саме там відбулася остання торгівля. Продаж працює аналогічно лише тому, що ви зараз працюєте на стороні ставок книги замовлень і, можливо, рухаєте книгу замовлень (і ціну) вниз. Іншими словами, розміщуючи замовлення на покупку та продаж,

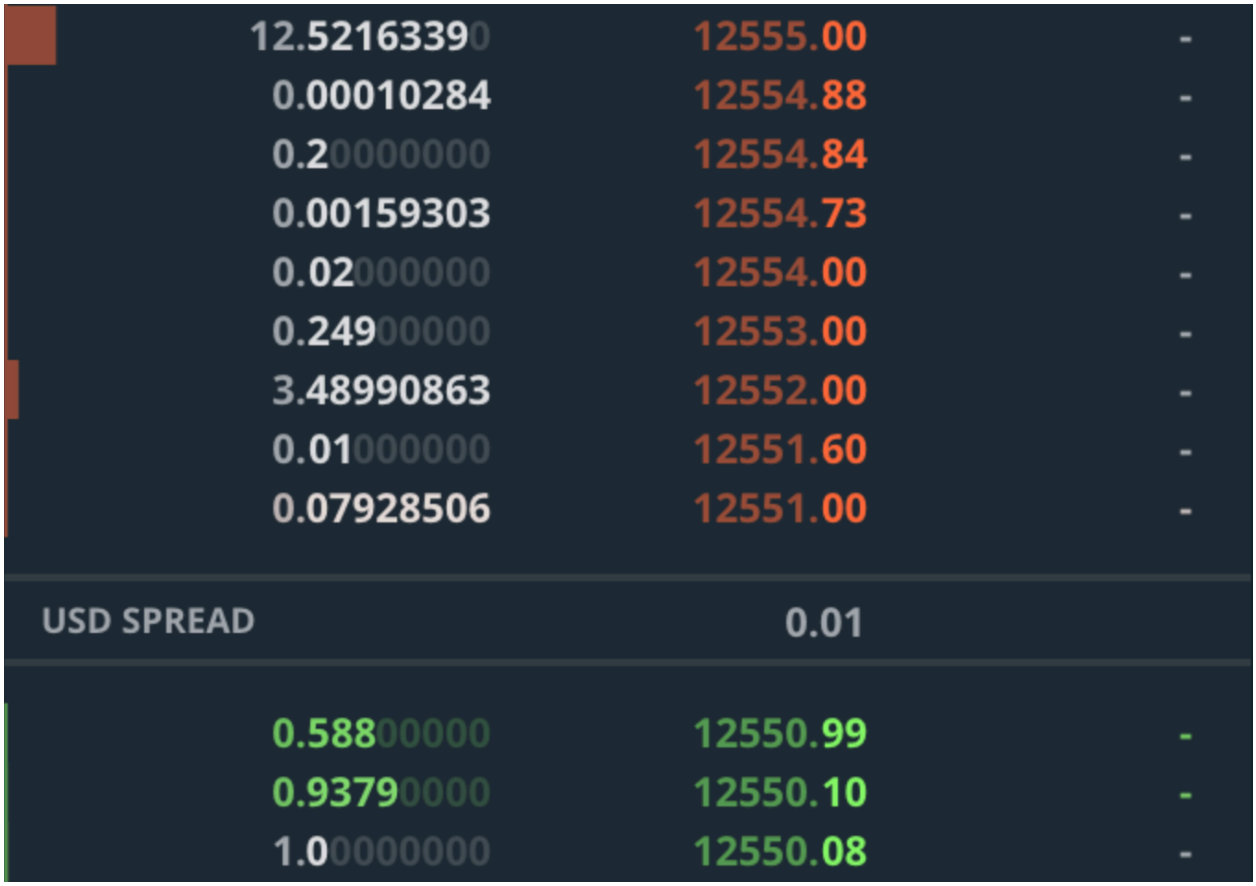

Рисунок 2.2 — Книга замовлень

ви вилучаєте обсяг з книги замовлень. Якщо ваші замовлення досить великі, ви можете змістити книгу замовлень на кілька рівнів. Насправді, якби ви розмістили дуже велике замовлення на кілька мільйонів доларів, ви змінили б замовлення та ціну значно.

Як замовлення потрапляють до книги замовлень? Це різниця між ринковими та лімітними замовленнями. У наведеному вище прикладі ви оформили ринкове замовлення, яке в основному означає ”Купівля / продаж X суми BTC за найкращою можливою ціною зараз”. Якщо ви не будете уважні до того, що є в книзі замовлень, ви можете заплатити значно більше, ніж показує поточна ціна. Наприклад, уявіть, що більшість нижчих рівнів у книзі замовлень мали лише обсяг 0,001 BTC. Більшість обсягу вашої покупки потім узгоджуватиметься на значно вищому, дорожчому, рівні цін. Якщо ви подаєте лімітне замовлення, яке також називається пасивним замовленням, ви вказуєте ціну та кількість, які ви хочете придбати чи продати. Замовлення буде розміщено в

книзі, і ви можете скасувати його до тих пір, поки воно не збігається. Наприклад, припустимо, що ціна на біткойн становить 10 000 доларів, але ви хочете продати за 10 1010 доларів. Ви розміщуєте лімітне замовлення. По-перше, нічого не відбувається. Якщо ціна продовжує рухатися вниз, ваше замовлення просто сидітиме там, нічого не робить, і ніколи не буде відповідати. Ви можете скасувати її будь-коли. Однак якщо ціна рухається вгору, ваше замовлення в якийсь момент стане найкращою ціною в книзі, і наступна особа, яка подає ринкове замовлення на достатню кількість, відповідатиме їй.

Ринкові замовлення беруть ліквідність з ринку. Порівнюючи замовлення з книги замовлень, ви знімаєте можливість торгувати з іншими людьми — залишилось менше обсягу! Ось чому ринкові замовлення або учасники ринку часто повинні платити більш високу плату, ніж виробники ринку, які кладуть замовлення в книгу. Обмежуйте замовлення, що забезпечують ліквідність, оскільки вони надають іншим можливість торгувати. У той же час лімітні замовлення гарантують, що ви не заплатите більше, ніж ціна, зазначена в лімітному порядку. Однак ви не знаєте, коли або якщо хтось відповість вашому замовленню. Ви також даєте ринку інформацію про те, якою ви вважаєте, якою повинна бути ціна. Це також може використовуватися для маніпулювання іншими учасниками ринку, які можуть діяти певним чином, виходячи з замовлень, які ви виконуєте чи додаєте до книги. Оскільки вони надають можливість торгувати та надавати інформацію, виробники ринку, як правило, платять менші збори, ніж суб’єкти ринку. Деякі біржі також надають зупинки, які дозволяють встановити максимальну ціну для ваших ринкових замовлень.

2.2 Дані

Для прикладу використовуються дані з криптовалютних бірж по причині їх відкритості і доступності. Багато бірж мають відкритий API.

- Торгівля (рис. 2.3). Сталася нова торгівля. Кожна торгівля має часову позначку, унікальний ідентифікатор, присвоєний біржею, ціною, розміром і стороною, як обговорювалося вище. Якби ви хотіли скласти

графік цін активу, ви просто складете ціну всіх торгів. Якщо ви хотіли скласти японські свічки, ви б відкрили вікно торгових подій протягом певного періоду, наприклад, п’яти хвилин, а потім склали графік вікон.


```
{
    "time": "2014-11-07T22:19:28.578544Z",
    "trade_id": 74,
    "price": "10.00000000",
    "size": "0.01000000",
    "side": "buy"
}
```


Рисунок 2.3 — Торгівля

- Оновлення книги замовлень (рис. 2.4). Оновлено один або кілька рівнів у книзі замовлень. Кожен рівень складається зі сторони (Buy = Bid, Sell = Ask), ціни / рівня та нової кількості на цьому рівні. Зауважте, що це зміни або дельти, і ви повинні створити повну книгу замовлень самостійно, об’єднавши їх.


```
{
    "type": "l2update",
    "product_id": "BTC-USD",
    "changes": [
        ["buy", "10000.00", "3"],
        ["sell", "10000.03", "1"],
        ["sell", "10000.04", "2"],
        ["sell", "10000.07", "0"]
    ]
}
```


Рисунок 2.4 — Оновлення книги замовлень

- Знімок книги замовлень (рис. 2.5). Схоже на оновлення книги замовлень, але знімок повної книги замовлень. Оскільки повна книга замовлень може бути дуже великою, скоріше і ефективніше використовувати події оновлення. Однак випадковий знімок може бути корисним.

Це майже все, що потрібно для ринкових даних. Потік вищезгаданих подій містить всю інформацію, яку ви бачили в інтерфейсі GUI. Ви можете

```
{
    "type": "snapshot",
    "product_id": "BTC-EUR",
    "bids": [["10000.00", "2"]],
    "asks": [["10000.02", "3"]]
}
```

Рисунок 2.5 — Знімок книги замовлень

уявити, як можна було зробити прогнозування на основі потоку вищевказаних подій.

2.3 Метрики для торгівлі

Розробляючи алгоритми торгівлі, для чого ви оптимізуєте? Очевидна відповідь — це прибуток, але це не вся історія. Вам також потрібно порівняти свою торгову стратегію з базовими і порівняти її ризик та мінливість з іншими інвестиціями. Ось кілька основних основних показників, якими користуються торговці.

- Net P&L (Чистий прибуток і збитки). Просто скільки грошей алгоритм заробляє (позитивно) або втрачає (негативно) протягом певного періоду часу за вирахуванням торгових комісій.
- Alpha та Beta. Альфа визначає, наскільки краще, з точки зору прибутку, ваша стратегія порівняно з альтернативною, відносно безризиковою, інвестицією, як державна облігація. Навіть якщо ваша стратегія вигідна, вам може бути краще інвестувати в безризикову альтернативу. Бета-версія тісно пов’язана і говорить про те, наскільки мінливою є ваша стратегія порівняно з ринковою. Наприклад, бета-версія 0,5 означає, що ваша інвестиція рухається на 1 долар, коли ринок рухається на 2 долари.
- Коефіцієнт Шарпа. Коефіцієнт Шарпа вимірює надлишкову віддачу на одиницю ризику, яку ви берете на себе. Це в основному ваша прибутковість капіталу над стандартним відхиленням, скоригованим на ризик.

Таким чином, чим вище, тим краще. Він враховує як мінливість вашої стратегії, так і альтернативну безризикову інвестицію.

- Максимальне зниження. Максимальне зниження - це максимальна різниця між локальним максимумом та наступним локальним мінімумом, ще одним показником ризику. Наприклад, максимальне скорочення 50% означає, що ви втрачаєте 50% свого капіталу в якийсь момент. Потім потрібно зробити 100% прибуток, щоб повернутися до початкової суми капіталу. Ясна річ, що нижче максимальне скорочення краще.
- Значення ризику (VaR). Значення ризику — це показник ризику, який кількісно визначає, скільки капіталу ви можете втратити за певний період часу з певною вірогідністю, припускаючи нормальні ринкові умови. Наприклад, 1-денний 5% VaR в розмірі 10% означає, що існує 5% шансів, що ви можете втратити більше 10% від інвестицій протягом дня.

2.4 Навчання з учителем

Перш ніж розглядати проблему з точки зору навчання з підкріпленням, давайте розберемося, як би ми йшли до створення вигідної торгової стратегії, використовуючи підхід навчання з учителем. Тоді ми побачимо, що в цьому не так, і чому ми хочемо використовувати методи навчання з підкріпленням.

Найбільш очевидний підхід, який ми можемо застосувати, — це прогнозування цін. Якщо ми можемо передбачити, що ринок підніметься, ми можемо купувати зараз і продавати, коли ринок перемістився. Або, що рівносильно, якщо ми прогнозуємо, що ринок знизиться, ми можемо піти на короткий термін (запозичити актив, який ми не маємо), а потім придбати, як тільки ринок переміститься. Однак з цим є кілька проблем.

Перш за все, яку ціну насправді прогнозуємо? Як ми вже бачили вище, у нас немає єдиної ціни. Остаточна ціна, яку ми сплачуємо, залежить від обсягу, доступного на різних рівнях книги замовлень, і від плати, яку нам потрібно сплатити. Наївна річ — передбачити середню ціну, яка є серединою між найкращою ставкою та найкращим запитом. Саме так роблять більшість

дослідників. Однак це лише теоретична ціна, а не те, на чому ми можемо виконувати замовлення, і може суттєво відрізнятися від реальної ціни, яку ми платимо.

Наступне питання — часовий масштаб. Чи прогнозуємо ми ціну наступної торгівлі? Ціна на наступну секунду? Хвилину? Годину? День? Інтуїтивно зрозуміло, чим далі в майбутньому ми хочемо прогнозувати, тим більше невизначеності і тим складніше стає проблема передбачення.

Давайте подивимось на приклад. Припустимо, що ціна BTC становить 10 000 доларів, і ми можемо точно передбачити, що ціна зросте з 10 000 до 10,050 доларів у наступну хвилину. Отже, чи означає це, що ви можете отримати 50 доларів прибутку, купуючи і продаючи? Давайте зрозуміємо, чому це не так:

- Ми купуємо, коли найкраща пропозиція — 10 000 доларів. Швидше за все, ми не зможемо заповнити всі наші 1,0 BTC за цією ціною, оскільки книга замовлення не має необхідного обсягу. Ми можемо змушені купувати 0,5 BTC за 10 000 доларів і 0,5 BTC за 10 1010 доларів, середня ціна 10 005 доларів. У GDAX ми також сплачуємо 0,3% комісійного збору, що відповідає приблизно 30 доларів.
- Ціна зараз становить 10,050 доларів, як було передбачено. Ми розміщуємо замовлення на продаж. Оскільки ринок рухається дуже швидко, до моменту доставки замовлення по мережі ціна вже знизилася. Скажімо, зараз це $10,045. Як і вище, ми, швидше за все, не можемо продати всі ваші 1 BTC за ціною. Можливо, ми змушені продавати 0,5 BTC — 10,045 доларів і 0,5 BTC за 10,040 доларів, середня ціна 10,042,5 долара. Тоді ми сплачуємо ще 0,3% комісійного збору, що відповідає приблизно 30 доларів.

Отже, скільки грошей ми заробили? $-10005-30-30+10042,5=-22,5$ дол. Замість того, щоб заробити $50, ми втратили $22,5, хоча ми точно передбачили великий рух цін протягом наступної хвилини! У наведеному вище прикладі для цього було три причини: відсутність ліквідності на рівні замовлень найкращого порядку, мережеві затримки та збори, жодна з яких контрольована модель не могла враховувати.

Які висновки тут можна зробити? Для того, щоб заробити гроші за допомогою простої стратегії прогнозування цін, ми повинні передбачити відносно великі зміни цін протягом більш тривалих періодів часу або бути дуже розумними щодо наших зборів та управління замовленнями. І це дуже складна проблема передбачення. Ми могли економити на зборах, використовуючи ліміт замість ринкових замовлень, але тоді у нас не було би гарантій щодо відповідності наших замовлень, і нам потрібно було б побудувати складну систему управління замовленнями та скасування замовлень.

Але є ще одна проблема з навчанням з учителем: у нас немає політики. У наведеному вище прикладі ми купили, тому що ми передбачили, що ціна рухатиметься вгору, і вона фактично рухається вгору. Все йшло за планом. Але що робити, якщо ціна знизилася? Ви б продали? Витримали позицію і чекали? Що робити, якщо ціна трохи піднялася вгору, а потім знову знизилася? Що робити, якщо ми не були впевнені в прогнозі, наприклад, на 65% і на 35% вниз? Ви б все-таки купили? Як ви обираєте поріг для розміщення замовлення?

Таким чином, вам потрібно більше, ніж просто модель прогнозування цін (якщо тільки ваша модель не є надзвичайно точною та надійною). Нам також потрібна політика на основі правил, яка враховує ваші прогнози цін і вирішує, що насправді робити: розміщуйте замовлення, не робіть нічого, скасовуйте замовлення тощо. Як ми можемо придумати таку політику? Як ми оптимізуємо параметри політики та пороги прийняття рішень? Відповідь на це не очевидна, і багато людей використовують просту евристику чи людську інтуїцію.

2.5 Типова стратегія

На щастя, існують рішення багатьох перерахованих вище проблем. Погана новина в тому, що рішення не дуже ефективні. Давайте розглянемо типовий робочий процес для розробки торгової стратегії (рис. 2.7):

а) Аналіз даних: Ви здійснюєте дослідницький аналіз даних, щоб знайти можливості торгівлі. Ви можете переглянути різні діаграми, обчислити

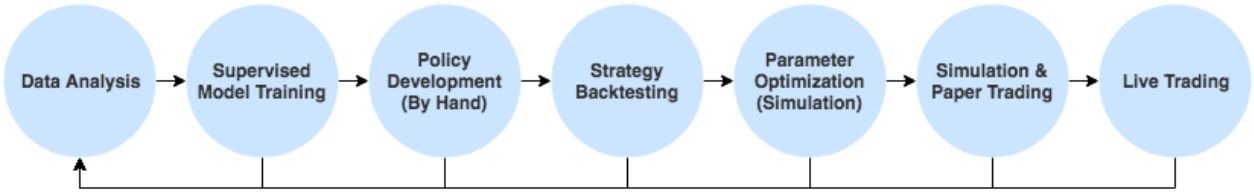

Рисунок 2.6 — Типова схема розробки політики

статистику даних тощо. Результатом цього кроку є ”ідея”для торгової стратегії, яку слід підтвердити;

б) Навчання з учителем: Якщо необхідно, ви можете навчити одну або кілька моделей навчання з учителем, щоб передбачити кількість цікавих, необхідних для роботи стратегії. Наприклад, прогнозування цін, прогнозування кількості тощо;

в) Розробка політики: Потім ви розробляєте політику, засновану на правилах, яка визначає, які дії потрібно вжити на основі поточного стану ринку та результатів моделей, що контролюються. Зауважте, що ця політика також може мати параметри, наприклад пороги прийняття рішень, які потрібно оптимізувати. Ця оптимізація робиться пізніше;

г) Backtesting: Ви використовуєте симулятор для тестування початкової версії стратегії на сукупності історичних даних. Симулятор може враховувати такі речі, як ліквідність замовлень, затримки в мережі, плату тощо. Якщо стратегія справно працює, ми можемо перейти до оптимізації параметрів;

д) Оптимізація параметрів: Тепер ви можете здійснювати пошук, наприклад, пошук по сітці, за можливими значеннями параметрів стратегії, таких як пороги або коефіцієнт, знову ж таки, використовуючи тренажер та набір історичних даних. Тут перевищення історичних даних є великим ризиком, і ви повинні бути обережними з використанням правильних наборів перевірки та тестів;

е) Моделювання та торгівля папером: Перед тим, як стратегія розпочнеться реалізація, моделювання проводиться на нових ринкових даних у режимі реального часу. Це називається торгівлею на папері і допомагає

запобігти надмірному розміщенню. Тільки якщо стратегія є успішною в торгівлі папером, вона розгортається в реальному середовищі;

ж) Жива торгівля: Стратегія зараз працює на біржі;

Це складний процес. Це може дещо відрізнятися залежно від фірми чи дослідника, але щось подібне відбувається, коли розробляються нові торгові стратегії. Тепер, чому цей процес не є ефективним? Причин є кілька:

а) Цикли ітерації повільні. Крок 1-3 в основному заснований на інтуїції, і ви не знаєте, чи працює ваша стратегія, поки не буде здійснена оптимізація на кроці 4-5, можливо, змусивши вас почати з нуля. Насправді кожен крок пов'язаний з ризиком невдачі і змушує вас почати з нуля;

б) Моделювання приходить занадто пізно. Ви чітко не враховуєте факторів навколишнього середовища, таких як затримки, збори та ліквідність до кроку 4. Чи не повинні ці речі безпосередньо інформувати про розробку стратегії чи параметри вашої моделі?

в) Політика розробляється незалежно від моделей, хоча вони тісно взаємодіють. Контрольовані прогнози є вкладом у політику. Чи не було б сенсу спільно їх оптимізувати?

г) Політика проста. Вони обмежуються тим, що люди можуть придумати;

д) Оптимізація параметрів неефективна. Наприклад, припустимо, що ви оптимізуєте комбінацію прибутку та ризику, і ви хочете знайти параметри, які дають вам високу коефіцієнт різкості. Замість використання ефективного підходу, заснованого на градієнті, ви робите неефективний пошук по сітці та сподіваєтесь, що ви знайдете щось хороше (при цьому не перестарайтеся).

Тепер розглянемо як з цією задачею впорається навчання з підкріпленням.

2.6 Використання навчання з підкріпленням для торгівлі

Традиційну проблему навчання з підкріпленням можна сформулювати як процес прийняття рішень Маркова (MDP) (рис. **??**). У нас є агент, що діє в оточенні. Кожен момент часу t агент отримує як вхід поточний стан, S_t, здійс-

нює дію A_t і отримує винагороду R_{t+1} і наступний стан S_{t+1}. Агент вибирає дію на основі деякої політики $\pi : A_t = \pi(S_t)$. Наша мета - знайти політику, яка максимально збільшує сукупну винагороду $\sum R_t$ протягом деякого кінцевого або нескінченного часового горизонту.

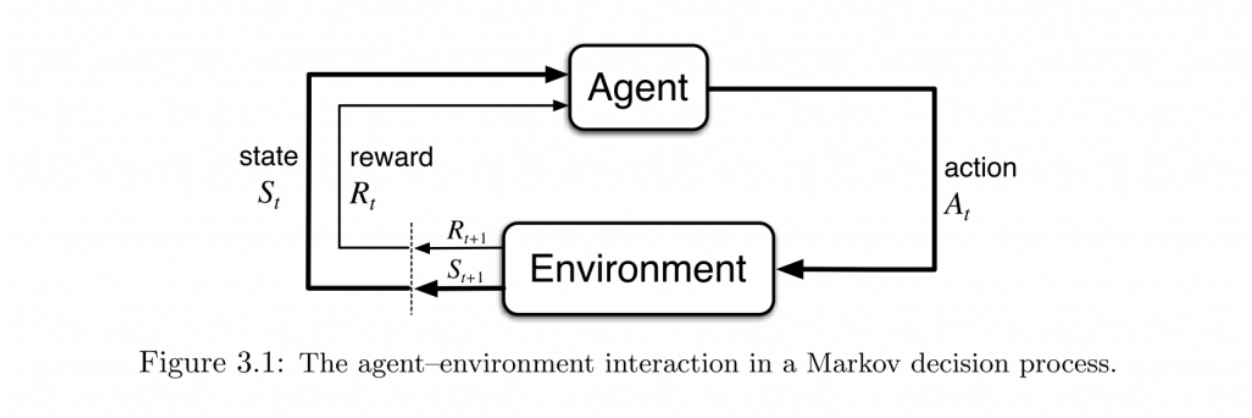

Figure 3.1: The agent–environment interaction in a Markov decision process.

Рисунок 2.7 — Взаємодія агента і середовища у Марковському процесі

2.6.1 Агент

Почнемо з легкої частини. Агент — наш торговий агент. Ви можете думати про агента як про людину-торговця, який відкриває графічний інтерфейс біржі та приймає торгове рішення, виходячи з поточного стану біржі та його рахунку.

2.6.2 Середовище

Це вже дещо складніше. Очевидною відповіддю було б те, що обмін — це наше середовище. Але важливо зазначити, що на одній біржі є багато інших агентів, як людей, так і алгоритмічних гравців ринку. Припустимо на мить, що ми вживаємо заходів щомісяця (докладніше про це нижче). Ми вживаємо певних дій, почекаємо хвилину, отримуємо новий стан, вживаємо ще одну дію тощо. Коли ми спостерігатимемо новий стан, це буде реакція ринкового середовища, яка включає відповідь інших агентів. Таким чином, з точки

зору нашого агента, ці агенти також є частиною навколишнього середовища. Вони не те, що ми можемо контролювати.

Однак, складаючи інших агентів у якесь велике складне середовище, ми втрачаємо можливість явно моделювати їх. Наприклад, можна уявити, що ми могли б навчитися створювати алгоритми та стратегії, якими керують інші торговці, а потім навчитися їх використовувати. Це зробило б нас проблемою багатоагентним навчанням з підкріплення (MARL), яка є активною дослідницькою сферою. Для простоти, припустимо, ми не робимо цього, і припустимо, що ми взаємодіємо з єдиним складним середовищем, яке включає поведінку всіх інших агентів.

2.6.3 Стан

У випадку торгів на біржі ми не спостерігаємо повного стану навколишнього середовища. Наприклад, ми не знаємо про те, що інші агенти знаходяться в оточенні, скільки їх є, які залишки на їхніх рахунках або які їхні відкриті лімітні замовлення. Це означає, що ми маємо справу з частково спостережуваним процесом рішення Маркова (POMDP). Що зауважує агент — це не фактичний стан середовища довкілля, а деяка деривація цього. Назвемо спостереження X_t, яке обчислюється за допомогою деякої функції повного стану $X_t \sim O(S_t)$.

У нашому випадку спостереження на кожному кроці часу — це просто історія всіх обмінних подій (описаних у розділі даних вище), отриманих до часу t. Ця історія подій може бути використана для створення поточного стану обміну. Однак для того, щоб наш агент приймав рішення, слід спостерігати ще кілька речей, зокрема спостереження за поточним рахунком та замовлення на відкритий ліміт, якщо такі є.

2.6.4 Часовий масштаб

Нам потрібно вирішити, в якому часовому масштабі ми хочемо діяти. Дні? Години? Хвилини? Секунди? Наносекунди? До всього цього потрібні різні підходи. Хтось, хто купує актив і тримає його протягом декількох днів, тижнів або місяців, часто робить довгострокову ставку на основі аналізу, наприклад Чи буде Bitcoin успішним?. Найчастіше ці рішення визначаються зовнішніми подіями, новинами або фундаментальним розумінням вартості або потенціалу активів. Оскільки такий аналіз, як правило, вимагає розуміння того, як працює світ, автоматизувати за допомогою методів машинного навчання може бути важко. З іншого боку, у нас є високочастотні методи торгівлі (HFT), де рішення майже повністю ґрунтуються на сигналах мікроструктури ринку. Рішення приймаються на часових шкалах наносекунди, а торгові стратегії використовують спеціальні підключення до бірж та надзвичайно швидкі, але прості алгоритми роботи апаратного забезпечення FPGA. Ще один спосіб подумати над цими двома крайнощами — це поняття людство. Перший вимагає широкого перегляду картини та розуміння того, як працює світ, людської інтуїції та аналізу на високому рівні, а останній — про просте, але надзвичайно швидке узгодження шаблону.

Нейронні мережі користуються популярністю, оскільки, маючи багато даних, вони можуть вивчити складніші уявлення, ніж алгоритми, такі як лінійна регресія чи наївний байес. Але глибокі нейронні мережі також повільні, відносно кажучи. Вони не можуть робити прогнози на наносекундних масштабах часу і тому не можуть конкурувати зі швидкістю алгоритмів HFT. Ось чому я думаю, що солодке місце знаходиться десь посеред цих двох крайнощів. Ми хочемо діяти у часовій шкалі, коли ми можемо аналізувати дані швидше, ніж це можливо, але де розумніші дозволяють нам обіграти швидкі, але прості алгоритми. Торговці-люди, можуть також діяти в цих часових масштабах, але не так швидко, як алгоритми. І вони, звичайно, не можуть синтезувати ту саму кількість інформації, яку може алгоритм за той самий проміжок часу. Це наша перевага.

Ще однією причиною діяти у відносно короткі часові шкали є те, що

шаблони даних можуть бути більш очевидними. Наприклад, оскільки більшість торговців-людей дивляться на точно такі ж (обмежені) графічні інтерфейси користувача, які мають заздалегідь визначені ринкові сигнали (наприклад, сигнал MACD, вбудований у багато обмінних графічних інтерфейсів), їх дії обмежуються інформацією, присутньою в цих сигналах , що призводить до певних моделей дій. Аналогічно алгоритми, що працюють на ринку, діють на основі певних зразків. Ми сподіваємось, що алгоритми Deep RL можуть підібрати ці зразки та використовувати їх.

Зауважте, що ми можемо також діяти на змінних масштабах часу на основі деякого тригерного сигналу. Наприклад, ми могли вирішити вжити заходів, коли на ринку відбулася велика торгівля. Такий, як агент на основі тригера, все ще приблизно відповідатиме деякій часовій шкалі, залежно від частоти події тригера.

2.6.5 Простір дій

У навчанні з підкріпленням ми робимо розмежування між дискретними (кінцевими) та безперервними (нескінченними) просторами дій. Залежно від того, наскільки складним ми хочемо бути нашим агентом, ми маємо пару варіантів. Найпростішим підходом було б три дії: купувати, утримувати та продавати. Це працює, але це обмежує нас у розміщенні ринкових замовлень та вкладанні детермінованої кількості грошей на кожному кроці. Наступним рівнем складності було б дозволити нашому агенту дізнатися, скільки грошей інвестувати, наприклад, виходячи з невизначеності нашої моделі. Це поставило б нас у простір безперервної дії, оскільки нам потрібно визначитися як з (дискретною) дією, так і з (безперервною) величиною. Ще складніший сценарій виникає тоді, коли ми хочемо, щоб наш агент міг розміщувати лімітні замовлення. У такому випадку наш агент повинен визначити рівень (ціну) та кількість замовлення.

Висновки до розділу 2

В даному розділі проведено аналіз фінансових ринків та застосування навчання з підкріпленням до торгівлі на них. В межах цього аналізу було досліджено структуру ринку, принципи його роботи, основні критерії якості стратегії до торгівлі на фінансових ринках. Було запропоновано замінити довгий та складний процес побудови стратегії торгівлі з використанням навчання з учителем на рішення на основі навчання з підкріпленням, що позбавляє від необхідності в великій кількості роботи, яку необхідно проробити, якщо не використовувати навчання з підкріпленням.

РОЗДІЛ 3

ЕКСПЕРИМЕНТ

Розглянувши усі необхідні теоретичні деталі, перейдемо безпосередньо до побудови реальної моделі і її тестування.

3.1 Дані

Для перевірки рішення було обрано кілька акцій таких компаній, як Google, Apple, Tesla. В якості джерела інформацію про ціну тих чи інших акцій було обрано Yahoo Finance.

На рис. 3.1 зображено як виглядають дані.

Time Period: Dec. 01, 2018 - Dec. 01, 2019 Show: Historical Prices Frequency: Daily Apply

Currency in USD Download Data

Date	Open	High	Low	Close*	Adj Close**	Volume
Nov. 29, 2019	3,147.18	3,150.30	3,139.34	3,140.98	3,140.98	1,743,020,000
Nov. 27, 2019	3,145.49	3,154.26	3,143.41	3,153.63	3,153.63	3,033,090,000
Nov. 26, 2019	3,134.85	3,142.69	3,131.00	3,140.52	3,140.52	4,595,590,000
Nov. 25, 2019	3,117.44	3,133.83	3,117.44	3,133.64	3,133.64	3,511,530,000
Nov. 22, 2019	3,111.41	3,112.87	3,099.26	3,110.29	3,110.29	3,226,780,000
Nov. 21, 2019	3,108.49	3,110.11	3,094.55	3,103.54	3,103.54	3,720,560,000
Nov. 20, 2019	3,114.66	3,118.97	3,091.41	3,108.46	3,108.46	4,034,890,000
Nov. 19, 2019	3,127.45	3,127.64	3,113.47	3,120.18	3,120.18	3,590,070,000
Nov. 18, 2019	3,117.91	3,124.17	3,112.06	3,122.03	3,122.03	3,436,690,000

Рисунок 3.1 — Дані

Приклади цін на закритті зображено на рис. 3.2, рис. 3.3, рис. 3.4 та рис. 3.5

На кожному графіку добре видно моменти коли ціни на акції сильно падають, а потім за якийсь час відновлюються в позиціях і навіть міцнішають.

Рисунок 3.2 — Індекс S&P

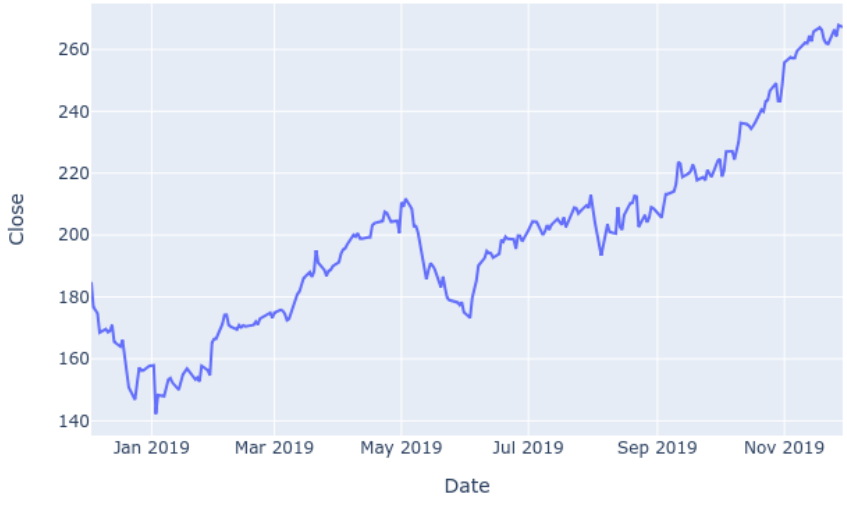

Рисунок 3.3 — Ціни на акції Apple

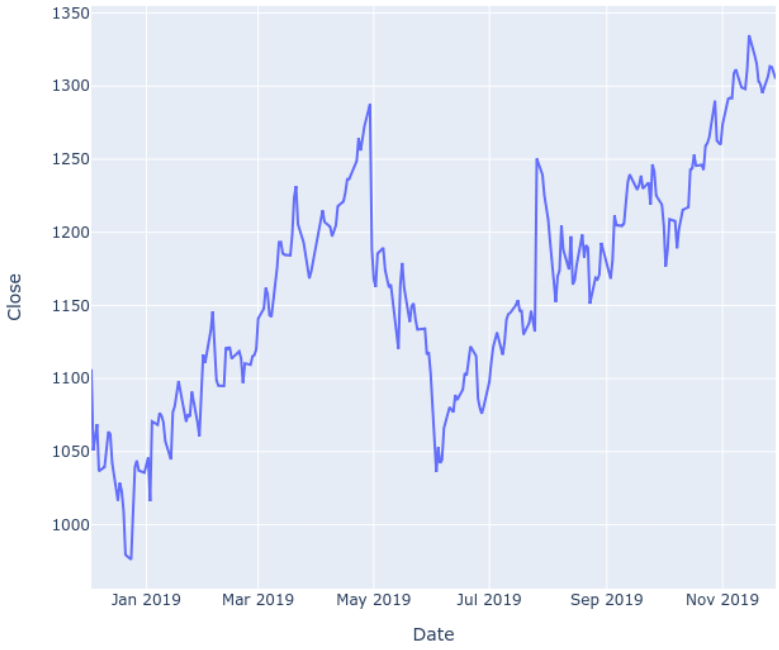

Рисунок 3.4 — Ціни на акції Google

Рисунок 3.5 — Ціни на акції Tesla

Самі в такі моменти нам би хотілось, щоб агент діяв — купив за максимально низьку ціну і продав за якомога більшу.

3.2 Архітектура моделі

Модель реалізує Q-навчання, яке було описане у першому розділі. У якості середовища були взяті ціни за 10 останніх днів, далі, за допомогою фреймворка Keras була побудована проста нейронна мережа, яка намагалась вивчити політику.

Модель реалізує дуже цікаву концепцію під назвою перепрогравання досвіду. Ця методика, що використовується у відомому AlphaGo, покращує стабільність моделі, зберігаючи попередній досвід агента та випадково відтворюючи їх.

Функція втрат, що мінімізувалась наступна:

$$loss = \left(r + \gamma \max_{a'} Q(s, a') - Q(s, a)\right)^2$$

В загальному, архітектура моделі зображено на рис. 3.6. У якості нейронної мережі була використана звичайна повнозв'язна трьох шарова нейронна мережа.

3.3 Результати

В силу незначних обчислювальних потужностей, мережа тренувалась не тривалий час, і при продовженні цього процесу є шанс, що результати будуть кращими.

На рис. 3.7 зображено результати роботи алгоритму на акціях Alibaba. Зелені крапочки символізують купівлю, червоні — продаж. В цілому видно розумну поведінку — на спаді модель надавала перевагу купівлі, а при зростанні ціни більше продавала. Але при цьому на тестовій ділянці, по закінченню роботи алгоритму ми опинились з 300 доларами менше, ніж на початку.

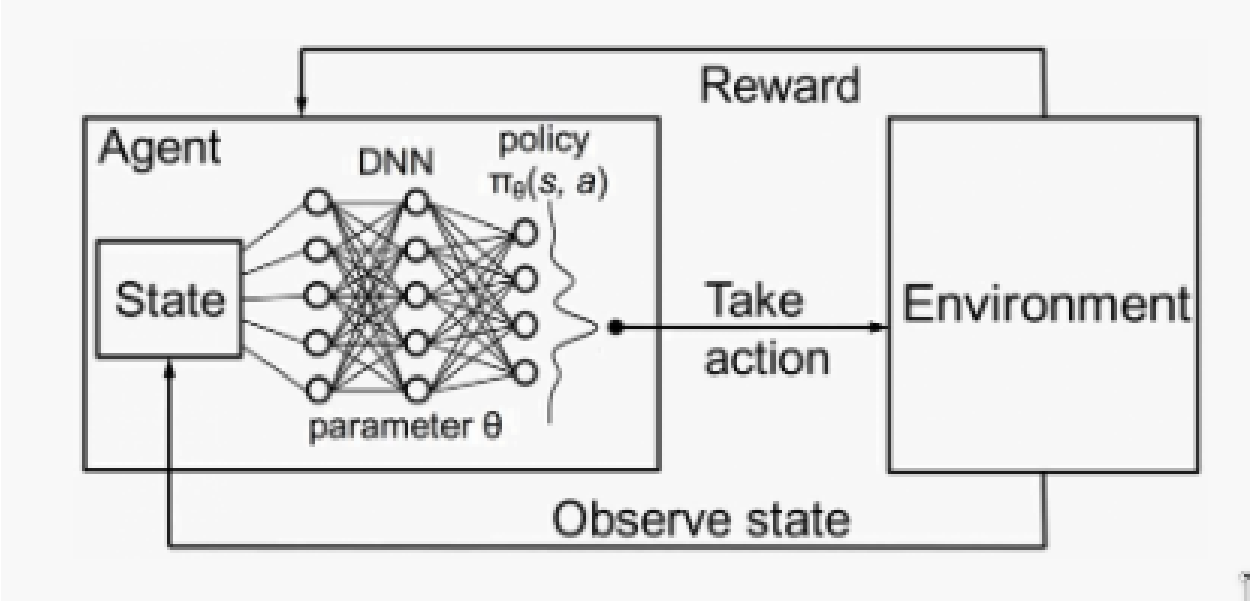

Рисунок 3.6 — Архітектура моделі

Скоріш за все, це обумовлено тим, що на початку модель занадто оптимістично скупила багато акцій, а по графіку ми бачимо, що до кінця періоду ціна впала сильно і не встигла піднятись до показників початку періоду.

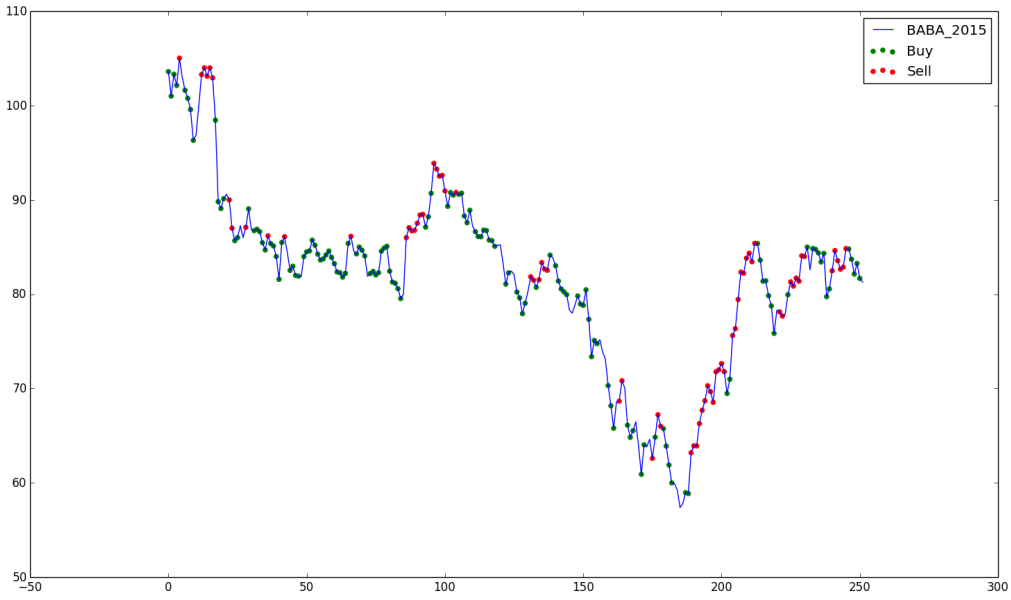

Рисунок 3.7 — Результати роботи на акціях Alibaba

На рис. 3.8 зображено результати роботи алгоритму на акціях Apple.

Позначки мають те саме значення. Поведінка дуже схожа на поведінку на попередньому прикладі. При цьому, в даному випадку ми «заробили» 160 доларів. Це обумовлено тим, що ціни на акції в цілому виросли за цей проміжок часу.

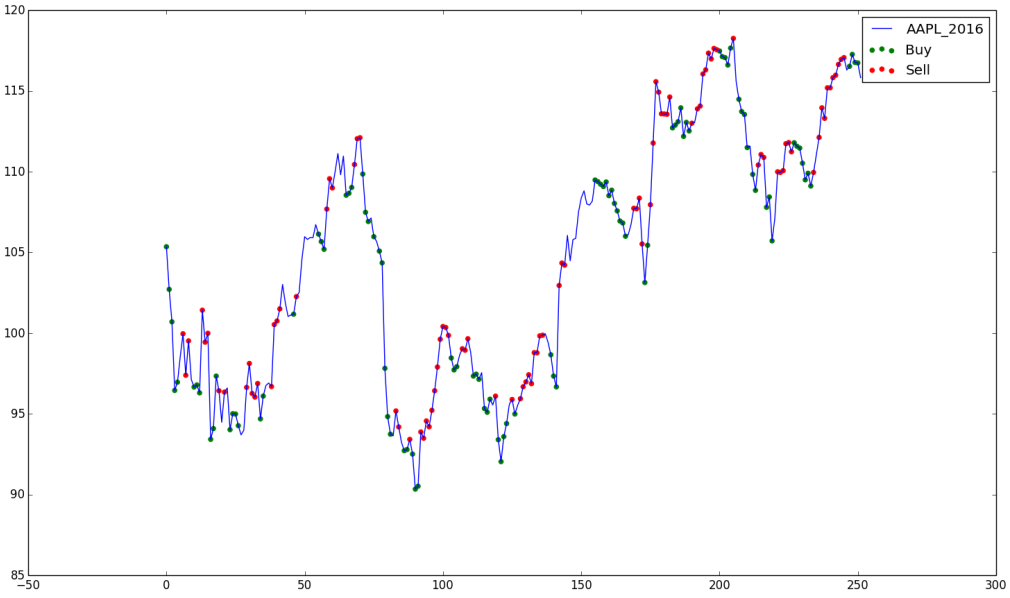

Рисунок 3.8 — Результати роботи на акціях Apple

Висновки до розділу 3

У цьому розділі представлені результати роботи реалізованого алгоритму. Отримані результати радше говорять про те, що алгоритм потрібно навчати довший час і не квапитись довіряти йому власні гроші.

РОЗДІЛ 4

РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

В останні роки набув великої популярності такий вид малого підприємництва як стартап.

Стартап-проект – є комерційним проектом, який знаходиться в стані розробки, або нещодавно вийшов на ринок. Характерною особливістю стартапу, що відрізняє його від малого бізнесу, є оригінальність та інновації, він не може бути копією вже реалізованих ідей. При цьому проект не обов'язково повинен бути масштабного характеру, головне, щоб він був креативним, а його завдання – спрощувати людям будь-які дії в їх повсякденному житті.

Наразі, з появою Інтернету та сучасних технологій, стало простіше заходити на ринок, знаходити інвесторів та споживачів. З'явилося набагато більше можливостей для розвитку свого проекту за кордоном, ніж раніше. Проте розробка стартапу є досить ризикованим завданням. Не всім вдається довести свій стартап-проект до ринкового впровадження. За статистикою успіху досягає лише 10-20% від усіх стартап-проектів.

Запуск стартапу передбачає цілий ряд обов'язкових дій, в межах яких визначають ринкові перспективи стартапу, графік розробки, принципи організації виробництва, заходи з залучення інвесторів та аналіз ризиків.

4.1 Опис ідеї проекту

У таблиці 4.1 подано зміст ідеї стартап-проекту, можливі напрямки застосування та основні вигоди, що може отримати користувач товару. У таблиці 4.2 визначені сильні, слабкі та нейтральні сторони проекту.

Таблиця 4.1 — Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Програмний продукт для автоматичної торгівлі цінними паперами	Фінансові біржі	Дозволяє користувачам з різним рівнем підготовки проводити необхідну попередню обробку даних для побудови прогнозуючої моделі, будувати модель автоматичної торгівлі та одержувати гроші дані на основі побудованої моделі

Таблиця 4.2 — Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			
		Мій проект	Super automatic trader	IBM smart trader	SAS Enterprise Trader
1.	Ціна	Низька	Середня	Висока	Висока
2.	Функціонал	Вузький	Вузький	Широкий	Широкий

Отже, з табл. 4.2 можна визначити, що ціна є сильною характеристикою для потенційного товару, а функціонал, зважаючи на напрямки застосування товару, є нейтральною властивістю.

4.2 Технологічний аудит ідеї проекту

За результатами аналізу таблиці 4.3 можна зробити висновок про можливість технологічної реалізації проекту.

Таблиця 4.3 — Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технології
1.	Програмний продукт для автоматичної торгівлі цінними паперами	Торгівля на основі навчання з підкріпленням	Наявна	Доступна
2.		Прогнозування на основі нейронної мережі каскадного типу	Наявна	Доступна
	(Обрана технологія реалізації ідеї проекту: прогнозування на основі методу логістичної регресії			

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Проведемо аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (табл. 4.4).

Таблиця 4.4 — Попередня характеристика потенційного ринку стартапу

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	100 000 ум.од
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	75%

За результатами аналізу таблиці 4.4 можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням.

Визначимо потенційні групи клієнтів, їх характеристики, та сформуємо орієнтовний перелік вимог до товару для кожної групи (табл. 4.5).

Таблиця 4.5 — Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці груп клієнтів	Вимоги споживачів
1	Прийняття рішення щодо здійснення фінансових операцій стосовно криптовалюти	Трейдингові компанії	Відмінність сфер діяльності клієнтів (торгівля в якості інвестора, трейдингова компанія)	Висока точність прогнозування. Простий у використанні. Швидкодія при обробці значного об’єму інформації

Проведемо аналіз ринкового середовища: таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. №№ 4.6-4.7).

Таблиця 4.6 — Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Наявність великої конкуренції	Вихід на ринок великої компанії	Вихід з ринку. Обрати нову цільову аудиторію. Передбачити переваги продукту, щоб повідомити про них саме після виходу великої компанії на ринок
2	Зміна потреб користувачів	Користувачам необхідні рішення з іншим функціоналом	Передбачити можливість додавання нового функціоналу до продукту

Таблиця 4.7 — Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Відсутність конкуренції	Відсутність аналогічних продуктів для користувача на вітчизняному ринку	Локалізація та адаптація сервісу для локальних груп. Адаптація до вітчизняних особливостей
2	Поява нових цільових груп клієнтів	Потреба в аналогічному продукті в інших сферах діяльності	Адаптація продукту під нові сфери використанняу

Проведемо аналіз пропозиції: визначимо загальні риси конкуренції на ринку (табл. 4.8).

Таблиця 4.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополістична	Існує декілька фірм-конкурентів	Підтримка якості продукту та постійні вдосконалення
2. За рівнем конкурентної боротьби - інтернаціональний	Фірми конкуренти з різних країн	Підтримувати продукт на національному ринку
3. За галузевою ознакою - внутрішньогалузева	Продукт використовується в одній галузі	Вдосконалювати продукт для застосування в інших галузях
4. Конкуренція за видами товарів: - товарно-родова	Присутня конкуренція з боку товарів-замінників	Розширювати функціонал продукту
5.За характером конкурентних переваг - неціна	Вдосконалення якості продукції, технології виробництва, інновацій	Випускати нові товари, які принципово відрізняються від своїх попередників та представляють модернізований варіант старої моделі
6. За інтенсивністю - немарочна	Роль торгової марки незначна	Приділяти увагу якості продукту а не бренду компанії

Після аналізу конкуренції проведемо більш детальний аналіз умов кон-

куренції в галузі (за моделлю 5 сил М. Портера) (табл. 4.9).

Таблиця 4.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Smart trader, IBM trader	APPBTC	Диференціація витрат, розширення каналів збуту	Контроль якості продукту	Наявність більш широкого функціоналу , зручнішого інтерфейсу
Висновки:	Середня конкурентна боротьба з вже існуючими на ринку гравцями	Є можливості виходу на ринок, але є і конкуренти. Строки – пів року	Постачальники не диктують умови роботи	Клієнти диктують умови роботи на ринку	Обмеження для роботи на ринку через товари замінники

На основі аналізу конкуренції (табл. 4.9), а також із урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.5) та факторів маркетингового середовища (табл.№№4.6-4.7) визначимо та обґрунтуємо перелік факторів конкурентоспроможності (табл. 4.10).

Таблиця 4.10 — Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Ціна	Більш доступна ціна збільшує кількість потенційних клієнтів
2	Функціонал	Функціонал направлений на предметну область
3	Зручний інтерфейс	Зручний інтерфейс робить продукт більш привабливим для клієнтів

За визначеними факторами конкурентоспроможності (табл. 4.10) проведемо аналіз сильних та слабких сторін стартап-проекту (табл. 4.11).

Таблиця 4.11 — Порівняльний аналіз сильних та слабких сторін «SmartTrade»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у Бали порівнянні з “SmartTrade”						
			-3	-2	-1	0	+1	+2	+3
1	Ціна	18		+					
2	Функціонал	10					+		
3	Зручний інтерфейс	12				+			

Складемо SWOT-аналіз (матриця аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities)) (табл. 4.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (табл. 4.11).

Таблиця 4.12 — SWOT-аналіз стартап-проекту

Сильні сторони: ціна, зручний інтерфейс	Слабкі сторони: функціонал
Можливості: Низька конкуренція, поява нових потреб споживачів	Загрози: Висока конкуренція, не відповідність потребам споживачів

На основі SWOT-аналізу визначимо альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (табл. 4.13).

Таблиця 4.13 — Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Створення програмного забезпечення	80%	3 місяці
2	Створення веб-сервісу	60%	5 місяці

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 4.14).

Таблиця 4.14 — Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Трейдингові компаніїї	Висока	Високий	Середня	Середня складність
2	Інші фінансові установи	Середня	Середній	Помірна	Висока складність
Які цільові групи обрано: 1					

Для роботи в обраних сегментах ринку сформуємо базову стратегію розвитку (табл. 4.15).

Таблиця 4.15 — Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку*
1	Надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів	Визначити потреби кожної з цільових груп, розробити стратегії приваблення споживачів та маркетингові комунікації	Оперативне реагування на зміни в ринковому попиті, орієнтованість на кінцевого споживача, висока якість продукту	Стратегія диференціації

Оберемо стратегію конкурентної поведінки (табл. 4.16).

Таблиця 4.16 — Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першо-прохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Не є першо-прохідцем	Шукати нових	Ні	Стратегія заняття конкурентної ніші

Сформуємо ринкову позицію, за якою споживачі мають ідентифікувати проект(табл. 4.17).

Таблиця 4.17 — Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Простий та зручний користувацький інтерфейс, надійність та безпека, швидкість роботи продукти	Стратегія диференціації	Позиція на основі порівняння продукту компанії з продуктами конкурентів. Відмінні особливості споживачів	Автоматизація робочих процесів, зниження кредитних ризиків, зниження навантаження та часу

4.5 Розроблення маркетингової програми стартап-проекту

У табл. 4.18 підсумуємо результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 — Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Автоматизація робочих процесів	Продукт автоматизує такі процеси, як обробка даних та прийняття рішення щодо видачі кредиту	Після впровадження продукту процес прийняття рішення щодо видачі кредиту стає автоматизованийу
2	Зменшення кредитних ризиків	Продукт зменшує кредитні ризики банківських установ	Висока точність прогнозування знижує кредитні ризики банківських установ
3	Зниження навантаження та часу	Продукт знижує навантаження на персонал банківських установ та зменшує час видачі кредиту	Персоналу банків не потрібно самостійно аналізувати великий об’єм даних, що знижує навантаження на прискорює роботу

Розроблена трирівнева маркетингова модель товару(табл. 4.19).

Таблиця 4.19 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Програмний продукт для прогнозування кредитоспроможності фізичних осіб. Повинен бути зручним, швидким та безпечним
II. Товар у реальному виконанні	Властивості/характеристики М/Нм Вр/Тх /Тл/Е/Ор
	1. Попередня обробка даних 2. Побудова скорингової моделі 3. Прогнозування кредитоспроможності
	Якість: проходження тестування
	Пакування: відсутнє
	Марка: “SmartTrade ”
III. Товар із підкріпленням	До продажу: відсутнє
	Після продажу: навчання персоналу, супровід, технічна підтримка
Вихідний код програмного продукту є закритим, та не передається клієнтам і третім особам. На програмний продукт оформлено авторське право	

Визначимо цінові межі, якими необхідно керуватись при встановленні ціни на товар (табл. 4.20).

Таблиця 4.20 — Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/-послугу
1	2500$	2000$	Високий рівень доходів	Базова покупка та впровадження: нижня межа - 1000$, верхня межа - 2000$.

Визначимо оптимальну систему збуту (табл. 4.21).

Таблиця 4.21 — Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товар	Глибина каналу збуту	Оптимальна система збуту
1	Цільові клієнти – банківські установи, які бажають впровадити у своїй роботі сучасні засоби,допоможуть автоматизувати робочі процеси. Вони цікавляться інноваційними рішеннями, відвідують тематичні семінари та конференції	Формування попиту і стимулювання збуту. Встановлення контактів із споживачами. Просування маркетингової інформації	Нульова або однорівнева (сервіс безпосередньо продається споживачам та через посередників)	Прямий канал збуту до споживача, мінімізувати витрати на додаткові канали збуту

Розроблена концепція маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22 — Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Цільові клієнти – банківські установи, що займаються кредитуванням фізичних осіб, і бажають автоматизувати процес видачі кредиту, та зменшити кількість неповернутих кредитів. Вони цікавляться інноваційними рішеннями, відвідують тематичні семінари та конференції	Конференції, форуми, новини у сфері інноваційних технологій, періодичні видання у професійних галузях	Позиція на основі порівняння продукту компанії з продуктами конкурентів. Відмінні особливості споживачів	- інформувати про новий продукт та його переваги; - сформувати сприятливу думку; - сформувати образ марки та її виробника у свідомості споживачів; - збільшити потік покупців	Зменшуємо кредитні ризики. Прискорюємо та автоматизуємо процес видачі кредитів

Висновки до розділу 4

В даному розділі проведено аналіз створення та виведення на ринок стартап-проекту на основі програмного продукту, який було розроблено в рамках магістерської дисертації. В межах цього аналізу було розроблено опис самої ідеї проекту, визначено загальні напрями використання товару, проаналізовано ринкові можливості щодо впровадження проекту, визначено відмінності від конкурентів та розроблено стратегію виходу на ринок. Узагальнюючи проведений аналіз, можна зазначити, що є можливість ринкової комерціалізації проекту. Наявний попит, динаміка ринку зростає. З огляду на потенційні групи клієнтів, а саме фінансові установи, та високий рівень конкурентоспроможності проекту, є достатні перспективи для впровадження стартапу. Отже, подальша імплементація проекту є доцільною.

ВИСНОВКИ

Дана робота присвячена аналізу, побудові та використанню навчання з підкріпленням для торгівлі на фінансових біржах.

Після ознайомлення з теоретичним матеріалом щодо суті проблематики та принципів роботи технології навчання з підкріпленням, основними статистичними та математичними методами алгоритмічної торгівлі та застосування методів навчання з підкріпленням до них, було побудовано систему для прийняття рішень щодо операцій із цінними паперами в фінансових установах.

В якості практичного прикладу застосування СППР, було розроблено програмний продукт з використанням технологій Python у середовищі розробки Jupyter Notebook. У даній системі було реалізовано модель навчання з підкріпленням для короткострокової торгівлі цінними паперами.

Отримані результати показують, що методи, розглянуті в роботі, показують різні результати в залежності від виду цінних паперів, якими ми торгуємо. Модель не здатна побачити в далекій перспективі падіння цін на активи, тому може втратити багато грошей, зробивши поганий вклад в далекій перспективі.

Результати магістерської дисертації:

а) запропоновано архітектуру системи підтримки прийняття рішень для торгівлі цінними паперами на фінансовій біржі;
б) розроблено програмний продукт для аналізу та обробки даних, побудови моделі навчання з підкріпленням для торгівлі цінними паперами;
в) розроблений ПП апробовано на акціях таких компаній, як Google, Apple, Tesla, Alibaba;
г) виконано порівняльний аналіз з іншими методами.

Подальшими напрямками роботи можуть бути питання, що стосуються:

а) вдосконалення розробленої архітектури;
б) реалізації методів для автоматизації процесу торгівлі із використанням можливостей відкритих API.

Розроблений програмний продукт показав результати, що говорять про те, що на коротких проміжках, модель робить правильні рішення, але у випадках коли ціни на акції будуть падати на великих проміжках часу, то вона не спроможна цього передбачити і втратить гроші. Таким чином, цю модель доречно використовувати у випадках коли ціни на акції на великих проміжках часу принаймні не падають.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bain A. *The Senses and the Intellect*, London: Parker, 1855. 215 p.

2. Thorndike E. L. *Animal Intelligence: An Experimental Study of the Associative Processes in Animals*, New York: Macmillan, 1898. 368 p.

3. Turing A. M. *Intelligent Machinery, A Heretical Theory, Lecture given at Oxford*, London: Oxford, 1948. 98 p.

4. Shannon C. *This Mouse Is Smarter Than You Are*, Popular Science, 1952. P. 99–101.

5. Minsky M. *Neural Nets and the Brain Model Problem* Princeton: Princeton University, 1954. 157 p.

6. Auer P., Cesa-Bianchi N., Fischer P. *Finite-Time Analysis of the Multiarmed Bandit Prob- lem*, Machine Learning, 2002, vol. 47, no. 2–3. P. 235–256.

7. Minsky M. *Steps Toward Artificial Intelligence*, Computers and Thought / New York: McGraw-Hill,, 1963. P. 406–450.

8. Howard R. A. *Dynamic Programming and Markov Processes*, Cambridge, MA: MIT Press, 1960. 231 p.

9. Sutton R. S., Barto A. G. *Reinforcement Learning: An Introduction*, Cambridge, MA: MIT Press, 1998. 298 p.

10. Watkins C. J. C. H., Dayan P. *Q-learning*, Machine Learning, 1992. P. 279–292.

11. Tesauro G. *Practical Issues in Temporal Difference Learning*, Machine Learning, 1992, vol. 8. P. 257–277.

12. Tesauro G. *Temporal Difference Learning and TD-Gammon*, Communications of the ACM, 1995, vol. 38, no. 3. P. 58–68.

13. V. Mnih et al. *Playing Atari With Deep Reinforcement Learning*, NIPS Deep Learning Workshop, 2013. P. 58-68

14.V. Mnih et al., *Human-Level Control through Deep Reinforcement Learning*, Nature, 2015, vol. 518, no. 7540. P. 529–533.

15.A. Nair et al. *Massively Parallel Methods for Deep Reinforcement Learning*, URL: http://arxiv.org/abs/1507.04296.

16. V. Mnih et al., *Asynchronous Methods for Deep Reinforcement Learning* , URL: http://arxiv.org/abs/1602.01783.

17. Николенко, С., Кадурин, А., & Архангельская, Е. *Глубокое обучение*, 2017, СпБ: Питер, 432 с.

18. Goodfellow, I., Bengio, Y., & Courville, A. *Deep learning*. MIT press., 511 p.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import numpy as np
import math

# prints formatted price
def formatPrice(n):
        return ("-$" if n < 0 else "$") + "{0:.2f}".format(abs(n))

# returns the vector containing stock data from a fixed file
def getStockDataVec(key):
         vec = []
         lines = open("data/" + key + ".csv", "r").read().splitlines()

         for line in lines[1:]:
                 vec.append(float(line.split(",")[4]))

        return vec

# returns the sigmoid
def sigmoid(x):
        return 1 / (1 + math.exp(-x))

# returns an an n-day state representation ending at time t
def getState(data, t, n):
        d = t - n + 1
        block = data[d:t + 1] if d >= 0 else -d * [data[0]] + data[0:t + 1] # pad with t0
        res = []
        for i in xrange(n - 1):
                res.append(sigmoid(block[i + 1] - block[i]))

        return np.array([res])

        from agent.agent import Agent
        from functions import *
        import sys

        if len(sys.argv) != 4:
                print "Usage: python train.py [stock] [window] [episodes]"
                exit()

        stock_name, window_size, episode_count = sys.argv[1], int(sys.argv[2]), int(sys.argv[3])

        agent = Agent(window_size)
        data = getStockDataVec(stock_name)
        l = len(data) - 1
        batch_size = 32

        for e in xrange(episode_count + 1):
                print "Episode " + str(e) + "/" + str(episode_count)
                state = getState(data, 0, window_size + 1)

                total_profit = 0
                agent.inventory = []
```

```
        for t in xrange(l):
                action = agent.act(state)

# sit
        next_state = getState(data, t + 1, window_size + 1)
        reward = 0

        if action == 1: # buy
                agent.inventory.append(data[t])
                print "Buy: " + formatPrice(data[t])

        elif action == 2 and len(agent.inventory) > 0: # sell
                bought_price = agent.inventory.pop(0)
                reward = max(data[t] - bought_price, 0)
                total_profit += data[t] - bought_price
                print "Sell: " + formatPrice(data[t]) + " | Profit: " + formatPrice(data[t]

        done = True if t == l - 1 else False
        agent.memory.append((state, action, reward, next_state, done))
        state = next_state

        if done:
                print "--------------------------------"
                print "Total Profit: " + formatPrice(total_profit)
                print "--------------------------------"

        if len(agent.memory) > batch_size:
                agent.expReplay(batch_size)

        if e % 10 == 0:
                agent.model.save("models/model_ep" + str(e))

        import keras
        from keras.models import load_model

        from agent.agent import Agent
        from functions import *
        import sys

        if len(sys.argv) != 3:
                print "Usage: python evaluate.py [stock] [model]"
                exit()

        stock_name, model_name = sys.argv[1], sys.argv[2]
        model = load_model("models/" + model_name)
        window_size = model.layers[0].input.shape.as_list()[1]

        agent = Agent(window_size, True, model_name)
        data = getStockDataVec(stock_name)
        l = len(data) - 1
        batch_size = 32

        state = getState(data, 0, window_size + 1)
        total_profit = 0
        agent.inventory = []

        for t in xrange(l):
```

```
            action = agent.act(state)

        # sit
            next_state = getState(data, t + 1, window_size + 1)
            reward = 0

            if action == 1: # buy
                agent.inventory.append(data[t])
                print "Buy: " + formatPrice(data[t])

            elif action == 2 and len(agent.inventory) > 0: # sell
                bought_price = agent.inventory.pop(0)
                reward = max(data[t] - bought_price, 0)
                total_profit += data[t] - bought_price
                print "Sell: " + formatPrice(data[t]) + " | Profit: " + formatPrice

            done = True if t == l - 1 else False
            agent.memory.append((state, action, reward, next_state, done))
            state = next_state

            if done:
                print "--------------------------------"
                print stock_name + " Total Profit: " + formatPrice(total_profit)
                print "--------------------------------"
```